

# NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



19980803 048

## THESIS

### FUEL-OPTIMAL LOW-EARTH-ORBIT MAINTENANCE

by

Karl E. Jensen

June, 1998

Thesis Advisors:

I. Michael Ross  
Fariba Fahroo

Approved for public release; distribution is unlimited.

DEIC QUALITY INSPECTED 1

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY <i>(Leave blank)</i>	2. REPORT DATE June 1998.	3. REPORT TYPE AND DATES COVERED Engineer's Thesis	
4. TITLE AND SUBTITLE: FUEL-OPTIMAL LOW-EARTH-ORBIT MAINTENANCE		5. FUNDING NUMBERS	
6. AUTHOR(S) Karl E. Jensen		8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT <i>(maximum 200 words)</i> First-order solutions indicate that a forced Keplerian trajectory (FKT) obtained by thrust-drag cancellation is as fuel-efficient as a Hohmann transfer. Further analysis has shown that the FKT is not Mayer-optimal. Therefore there must exist another trajectory that matches or exceeds the efficiency of the Hohmann transfer. The application of this result to the fuel-optimal orbit maintenance problem implies that periodic reboots must be more efficient than an FKT profile. This research begins with the formulation of an optimal periodic control (OPC) problem to determine the minimum fuel-reboost strategy. The problem is numerically solved by a spectral collocation method. The optimization code is further modified to increase accuracy and reduce sensitivity to initial guesses. The results of this effort identified a trajectory for a sample satellite that was 3.5% more efficient than an ideal impulsive Hohmann transfer over the same period of time. From the optimal code, a maximum thruster size is also identifiable for a set of initial conditions. The optimal trajectory can save as much as 10% of the propellant budget when compared to finite-burn Hohmann transfers.			
14. SUBJECT TERMS *Orbital Mechanics; Orbit Maintenance; Orbital Decay; Optimal Control Theory; Optimal Periodic Control Theory; Fuel-Optimal			15. NUMBER OF PAGES 129
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL



Approved for public release; distribution is unlimited.

**FUEL-OPTIMAL LOW-EARTH-ORBIT MAINTNENANCE**

Karl E. Jensen  
Lieutenant, United States Navy  
B.S., United States Naval Academy, 1990

Submitted in partial fulfillment  
of the requirements for the degree of

**AERONAUTICAL AND ASTRONAUTICAL ENGINEER  
MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**

**June 1998**

Author:

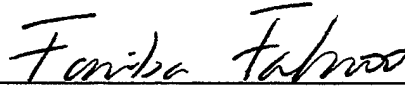


Karl E. Jensen

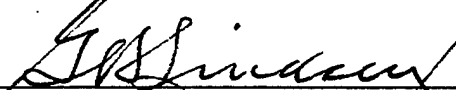
Approved by:



I. Michael Ross, Thesis Co-Advisor



Fariba Fahroo, Thesis Co-Advisor



G. Lindsey, Chairman

Department of Aeronautics and Astronautics



## ABSTRACT

First-order solutions indicate that a forced Keplerian trajectory (FKT) obtained by thrust-drag cancellation is as fuel-efficient as a Hohmann transfer. Further analysis has shown that the FKT is not Mayer-optimal. Therefore there must exist another trajectory that matches or exceeds the efficiency of the Hohmann transfer. The application of this result to the fuel-optimal orbit maintenance problem implies that periodic reboosts must be more efficient than an FKT profile. This research begins with the formulation of an optimal periodic control (OPC) problem to determine the minimum fuel-reboost strategy. The problem is numerically solved by a spectral collocation method. The optimization code is further modified to increase accuracy and reduce sensitivity to initial guesses. The results of this effort identified a trajectory for a sample satellite that was 3.5% more efficient than an ideal impulsive Hohmann transfer over the same period of time. From the optimal code, a maximum thruster size is also identifiable for a set of initial conditions. The optimal trajectory can save as much as 10% of the propellant budget when compared to finite-burn Hohmann transfers.



## TABLE OF CONTENTS

I. INTRODUCTION.....	1
II. ORBITAL MOTION AND OPTIMAL CONTROL.....	5
A. OPTIMAL CONTROL THEORY.....	5
B. EQUATIONS OF MOTION.....	6
C. NORMALIZATION.....	8
III. ORBIT RAISING MANEUVERS.....	13
A. THE LOW EARTH ATMOSPHERE.....	13
B. ORBIT RAISING MANEUVERS.....	15
IV. OPTIMAL PERIODIC CONTROL.....	21
A. PROBLEM FORMULATION.....	21
B. NUMERICAL METHOD.....	24
C. NONLINEAR PROGRAMMING CODE.....	27
1. Orbprop.....	28
2. Orbopt.....	29
3. Orbcrit.....	30
4. Other Programs.....	34
V. DISCUSSION OF RESULTS.....	35
A. INTRODUCTION.....	35
B. THE FREE CASE.....	35
C. FIXED TAU.....	46
D. BAND FIXED SIMULATION.....	56
VI. ANALYSIS OF OPTIMALITY.....	63
A. OPTIMALITY TEST CODES.....	63
B. VARIATION OF PARAMETERS.....	65
VII. CONCLUSION.....	81
LIST OF REFERENCES.....	83
APPENDIX: PROGRAM FILES.....	85
INITIAL DISTRIBUTION LIST.....	115





## ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my co-advisors, Professor I. Michael Ross and Professor Fariba Fahroo for their enthusiasm and perseverance in studying this problem. Their guidance and patience, from the problem's conception, to the multiple presentations, to the final culmination of this thesis is greatly appreciated. The success of this project is largely due to their support.



## LIST OF SYMBOLS

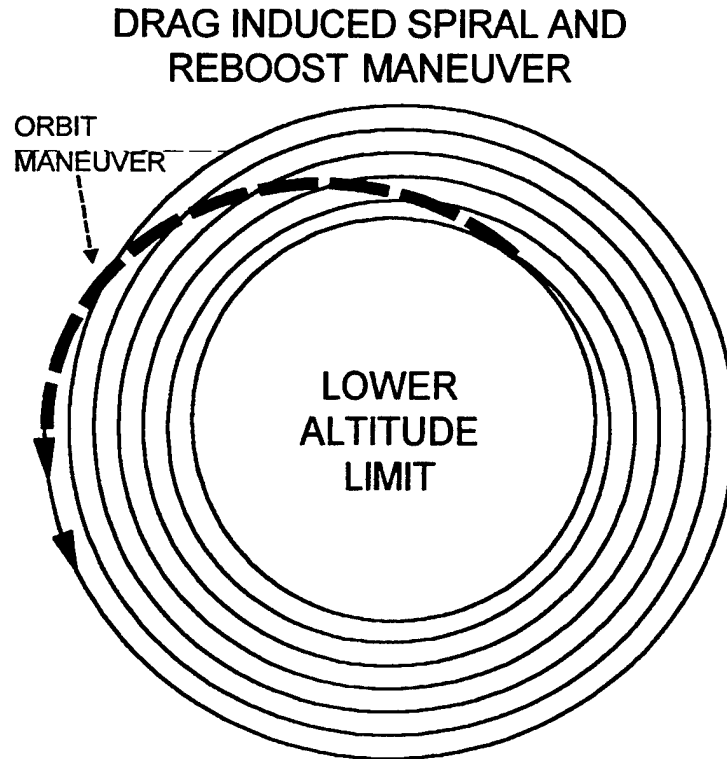
$r$	Radius
$V$	Velocity
$m$	Mass
$T$	Thrust
$v_e$	Exhaust Velocity
$g$	Acceleration due to Gravity
$C_D$	Drag Coefficient
$A$	Vehicle Surface Area, in the direction of motion
$B$	Ballistic Coefficient
$J$	Cost Function
$L$	Legendre Polynomial
$x$	State Vector
$u$	Control Vector
$D_n$	Differentiation Matrix
$w_k$	Weights
$\gamma$	Flight Path Angle
$\rho$	Atmospheric Density
$\theta$	Orbital Angle
$\alpha$	Control Angle
$\phi$	Lagrange Polynomial

## I. INTRODUCTION

The boundary of the Earth's atmosphere extends well into the operating area of low-Earth-orbiting satellites. As a vehicle circles the Earth at over seven kilometers per second, the few molecules that exist in the upper layers of the atmosphere are continually striking the vehicle's surface. Over time, these collisions decrease the orbital energy of the spacecraft, lowering its orbital altitude to an area filled with higher concentrations of molecules. This perturbing force is called atmospheric drag and is a large factor in determining the lifetime of any low-Earth-orbiting satellite. While the density of the atmosphere at altitudes where these satellites operate, 200 km to 500 km above the Earth's surface, is low ( $\sim 10^{-11}$  kg/m<sup>3</sup>), the velocity of the spacecraft is very high ( $\sim 8$  km/s). These two factors combine to produce a fairly significant drag force that operates in opposition to the velocity vector. A satellite that fails to counter these effects will continue to lose its orbital altitude, spiraling towards the Earth until it burns-up during reentry. To avoid this situation, mission lifetimes are extended by maneuvers that reboost the satellite to its original altitude periodically to prevent destructive atmospheric heating. This reboost maneuver is the largest source of propellant consumption for many low-Earth-orbiting vehicles and mission lifetime depends directly on the amount of fuel a spacecraft can carry for reboost. This thesis examines this process and attempts to maximize the satellite fuel efficiency by determining the optimal reboost trajectory through the use of optimal periodic control.

As a spacecraft circles the Earth, drag continually removes energy from the orbit. This causes an instantaneous reduction of the vehicle's orbital velocity. The change in velocity alters the original orbital shape, increasing the eccentricity. In a sense, the satellite enters into a "steeper" orbit that in turn increases the orbital velocity. This is known as the drag paradox, where the removal of orbital energy causes a satellite to travel faster. Since drag is higher at greater densities and velocities, the perturbing force of drag increases. The result is that the satellite follows a spiral path, which increases in velocity as the altitude decreases. Eventually drag will grow until significant atmospheric heating begins to occur. The reboost maneuver avoids a premature end to the satellite's lifetime by

reboosting it to a higher altitude, a region of reduced drag. Figure 1.1 shows this trajectory pattern and orbit transfer process.



*Figure 1-1*

The most common type of orbital transfer, mainly in terms of mission planning, is the Hohmann transfer. This maneuver begins with the satellite in a circular orbit at its lowest desired height. To reboost the vehicle, an impulsive burn is made which places the satellite into an elliptical transfer orbit. This transfer orbit has two important geometric properties. First, the perigee of this orbit is located at the position of the initial impulsive burn and secondly, the apogee of the orbit has an altitude that is equal to the desired final altitude. When the satellite reaches apogee, another impulsive burn is made to circularize the final orbit.

Previous work has examined the Hohmann transfer against other trajectories [Ref. 1,2]. Using first order estimates of different trajectories, optimal control theory states that the Hohmann transfer is not the most fuel-efficient solution [Ref. 3,4]. Thus there must exist a trajectory that maintains the satellite in a desired orbital band but with less consumed propellant. The previous thesis efforts at the Naval Postgraduate School have

examined several orbital transfer methods but did not surpass the efficiency of the Hohmann transfer. This effort takes a more rigorous approach in searching for the optimal solution to the orbit maintenance problem. Optimal periodic control (OPC) theory is applied to the system, which is governed by a set of equations of motion. Numerically solving the OPC problem gives a solution and a trajectory that can be adequately compared to the Hohmann transfer.

Instead of guessing a trajectory first and then comparing its performance against the Hohmann, this method mathematically derives an optimal trajectory that then can be compared. The goals of this thesis are twofold. First, this thesis will analyze the orbital transfer by numerically solving the OPC problem. Second, the thesis will identify a trajectory that is more fuel-efficient than currently used orbital transfer techniques.

Launch Vehicle	Cost
Space Shuttle	\$ 9,100
Atlas	\$ 13,900
Delta	\$ 15,300

*Figure 1-2*

The question becomes why is this method and research necessary and important. Conventional wisdom suggests that this problem has been solved. However, optimal control theory has demonstrated in many instances that the most intuitive solution is not necessarily the optimal solution. For example, to say that the shortest distance between two points is a straight line is not always true when one adds additional complexities or constraints, such as if the points are located on the surface of the Earth, to the system. Monetarily, this question should be examined in detail. Figure 1-2 shows the cost of raising a single kilogram of material into low-Earth orbit. Any savings at all over conventional transfer methods would soon add up to hundreds of thousands of dollars or more over the life of the spacecraft. The possibility of savings alone justifies this examination of optimal control theory applied to this problem.

Optimal control centers on the formulation and minimization of a cost function, in this case a function that describes the propellant expended. However, orbital motion is cyclic in nature. This means that the motion is periodic and repeats over a given time period. For many processes, periodic controls are more efficient than steady state operations. Acknowledging that fact, the problem then requires the examination of a further specialization of optimal control, namely, optimal periodic control. This process adds the complexity of periodic states and controls that allow a better description and representation of the periodic process. Through the use of numerical techniques, the optimal periodic control problem can be solved, creating a time history of states and controls that minimize the system's cost function.

The first step in examining this system and the orbit maintenance solution begins with the equations of motion, which are normalized for properly scaled numerical methods. Within the equations of motion, the state and control variables are identified. Boundary conditions are then determined with some of the conditions being a set of periodic functions. A spectral collocation method is used to numerically solve the optimal periodic control problem. This method seeks polynomial approximations for the states and controls in terms of their values at certain points or nodes. These nodes are called the Legendre-Gauss-Lobatto (LGL) points and are located at the roots of the first derivative of an  $n^{\text{th}}$ -degree Legendre polynomial. The performance measure for the periodic process is approximated by the creation of a cost function and referenced to a steady state solution. Through the use of the collocation method, the OPC problem is converted to a nonlinear programming problem that is solved using existing routines. The full optimization code is then optimized for quicker performance with more reliable results by introducing period constraints and eliminating certain variables. An example is tracked throughout the thesis to highlight the different aspects of the implementation of the OPC and the associated code for the orbit maintenance problem. Finally, the results of the optimization code are compared with existing trajectories and any fuel savings are noted.



## II. ORBITAL MOTION AND OPTIMAL CONTROL

### A. OPTIMAL CONTROL THEORY

For any problem that has an infinite number of solutions, further means must be used in order to choose the solution that maximizes the performance of the system at the lowest cost. The method designed to do this is called optimal control. The use of optimal control begins with a system that is described by a set of state variables, which describe its various physical limitations. Inputs into this system are called controls, which perturb the states within the system constraints over the time history of the problem. The performance of a set of controls versus another possible solution set is compared by a cost function that is minimized or maximized by the optimal control algorithm. The control set that minimizes the cost function is the optimal solution. Classical linear control systems generally relied upon "trial and error" processes that used numerous iterative techniques to determine the optimum set of controls. However, modern day problems require system dynamics that are too complex with many different performance criteria to be met, and classical methods are insufficient to solve these types of problems. The complexity of the problem is increased by another magnitude when periodicity is added. In the optimal periodic control problem, the states, controls, and perhaps most importantly the boundary functions can all be periodic. Orbital motion is a periodic problem. A spacecraft starts from a certain location, in this case defined by its velocity, radius, and flight path angle. After a time period that is either given or formulated by the control problem, the spacecraft must return to its original states. The process begins again continuing in a cyclic manner. The performance criterion used for this type of problem is the amount of fuel required over the problem's period,  $\tau$ .

## B. EQUATIONS OF MOTION

To first attack the optimal control problem the equations of motion for orbital flight must be obtained in order to mathematically model the physical system. The objective of the modeling process is to create the simplest mathematical description of the system to predict its response to any given input. The states are governed by a system of first order differential equations.

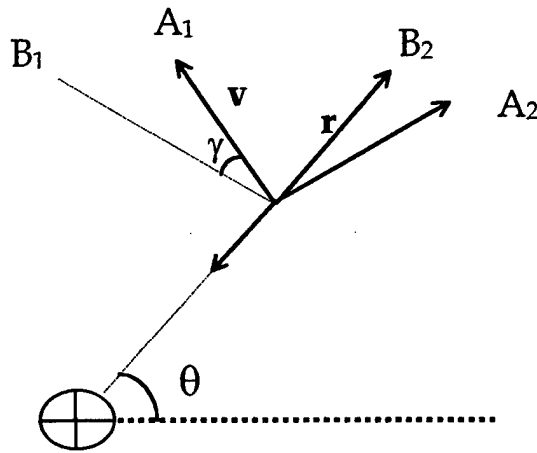


Figure 2-1

In inertial space ( $N$ ):

$$\begin{aligned} {}^N \dot{\vec{r}} &= \vec{v} \\ \text{and} & \\ {}^N \dot{\vec{v}} &= \frac{\vec{F}}{m} \end{aligned} \quad (1)$$

To find a differential expression for the radius, state  $r$ , the derivation begins by examining the radius in terms of its local body coordinates.

$$\vec{r} = r \hat{B}_2 \quad (2)$$

Taking the derivative and noting that  $\dot{\hat{B}}_2 = 0$ , equation (2) becomes:

$${}^B \dot{\vec{r}} = \dot{r} \hat{B}_2 \quad (3)$$

To transform the expression into the inertial coordinate system the transportation theorem is used and equation (3) becomes:

$${}^N \dot{\vec{r}} = {}^B \dot{\vec{r}} + {}^N \omega^B \times \vec{r} = \vec{v} \quad (4)$$

Simplifying:

$$\vec{v} = \dot{r} \hat{B}_2 + r \dot{\theta} \hat{B}_1 \quad (5)$$

From Figure 2.1 another expression for  $\vec{v}$  can be obtained.

$$\vec{v} = v \cdot \cos(\gamma) \hat{B}_1 + v \cdot \sin(\gamma) \hat{B}_2 \quad (6)$$

Since the expressions for  $\vec{v}$ , equations (5) and (6), are equal, the individual components along each body axis are also equal. An expression for  $\dot{r}$  can now be determined.

$$\dot{r} = v \cdot \sin(\gamma) \quad (7)$$

The derivation for  $\dot{v}$  follows the same general procedure as above but is worth some examination. The vector  $\vec{v}$  is defined in its own local coordinate system, frame  $A$  from Figure 2.1.  $\hat{A}_1$  is the unit vector in the direction of the velocity vector while  $\hat{A}_2$  is in the normal direction. Taking the derivative in the  $A$  frame:

$${}^A \dot{\vec{v}} = \dot{v} \hat{A}_1 \quad (8)$$

Using the transportation theorem to put  $\dot{\vec{v}}$  in terms of the inertial coordinate system, equation (8) becomes:

$${}^N \dot{\vec{v}} = \dot{v} \hat{A}_1 + {}^N \omega^A \times \vec{v} \quad (9)$$

From equation (1) and the summation of forces from Figure 2.1:

$${}^N \dot{\vec{v}} = \frac{\vec{F}}{m} = \frac{T \cos(\alpha) - D}{m} \cdot \hat{A}_1 - g \cdot \sin(\gamma) \cdot \hat{A}_1 - g \cdot \cos(\gamma) \cdot \hat{A}_2 + \frac{T \cdot \sin(\alpha)}{m} \cdot \hat{A}_2 \quad (10)$$

Equations (9) and (10) combine to give an expression for  $\dot{v}$ .

$$\dot{v} = \frac{T \cdot \cos(\alpha) - D}{m} - g \cdot \sin(\gamma) \quad (11)$$

The above expressions for  $\dot{r}$  and  $\dot{v}$  are two of the five states needed to adequately describe orbital motion. The other states are derived in a similar manner. The full set of equations of motion become:

$$\dot{r} = v \cdot \sin(\gamma) \quad (7)$$

$$\dot{v} = \frac{T \cdot \cos(\alpha) - D}{m} - g \cdot \sin(\gamma) \quad (11)$$

$$\dot{\gamma} = \left( \frac{v^2}{r} - g \right) \cdot \frac{\cos(\gamma)}{v} + \frac{T \cdot \sin(\alpha)}{m \cdot v} \quad (12)$$

$$\dot{m} = \frac{-T}{v_e} \quad (13)$$

$$\dot{\theta} = \frac{v}{r} \cdot \cos(\gamma) \quad (14)$$

### C. NORMALIZATION

The optimal periodic control problem is not solvable by analytical means and must be solved numerically. A numerical nonlinear code is used to solve for the states and controls and determine the best solution. The size of the physical constants involved with the problem of orbital motion varies greatly. For example, the typical atmospheric density at the altitudes in question lies approximately in the  $10^{-13}$  kg/m<sup>3</sup> range while the orbital velocity at the same altitudes is around  $10^5$  m/s. The numerical disparity between these two numbers poses significant computational difficulty for any code. Equation (11) which is the first order differential of the velocity includes the drag term,  $D$ , which includes atmospheric density. To better control the scaling of the physical constants, non-dimensional units are derived through the process of normalization.

For practical reasons, all numerical calculations are based upon reference values taken at a specific value in space. One of the popular ways of normalization in orbital motion is the creation of the canonical unit system. For Earth based systems, canonical units are referenced with large measurable Earth constants. One distance unit (DU) is equal to the radius of the Earth (6378 km). A velocity unit (VU) is defined as the orbital velocity of an imaginary object that circles the Earth at a distance of one DU, or simply at the surface of the Earth. A time unit (TU) is the amount of time it takes the same object to travel one radian about the orbit's center. The gravitational parameter,  $\mu$ , turns out to be one DU<sup>3</sup>/TU<sup>2</sup> or simply have a numerical value of one.

While canonical units are convenient in describing earth systems, a slight modification of the reference values decreases the level of numerical complexity for this problem. The

distance unit is referenced to the altitude where the satellite operates instead of the Earth's surface.

$$\bar{r} = \frac{r}{r_{ref}} \quad (15)$$

Numerically the code uses an altitude of 300 km for the basis of all radius calculations. In this case,  $r_{ref}$  becomes approximately 6678 km. When the vehicle is at an altitude of 300 km above the Earth's surface the value of  $\bar{r}$  is one.

Dividing the numerical value by the orbital velocity at the vehicle's reference altitude normalizes the velocity parameter.

$$\bar{v} = \frac{v}{v_{ref}} = \frac{v}{\sqrt{\frac{\mu}{r_{ref}}}} \quad (16)$$

Similarly mass and time can be normalized.

$$\bar{m} = \frac{m}{m_{ref}} \quad (17)$$

$$\bar{t} = \frac{t}{t_{ref}} \quad \text{where} \quad t_{ref} = \frac{r_{ref}}{v_{ref}} = \frac{r_{ref}}{\sqrt{\frac{\mu}{r_{ref}}}} \quad (18)$$

The first order differential equation for the radius can now be fully described with non-dimensional units. Equations (15) through (18) combine to allow the following derivation:

$$\dot{r} = \frac{dr}{dt} = v \cdot \sin(\gamma) \quad (7)$$

Substituting the reference values:

$$\frac{d\bar{r}}{d\bar{t}} = \frac{t_{ref} v_{ref}}{r_{ref}} \cdot \bar{v} \cdot \sin(\gamma) \quad (19)$$

$$\dot{\bar{r}} = 1 \cdot \bar{v} \cdot \sin(\gamma) \quad (20)$$

The structure of the first order differential does not change due to normalization, rather the numerical values must be reinterpreted as normalized values. The normalization of the differential velocity follows the same procedure but requires an extra referencing of thrust and drag.

$$\dot{\bar{v}} = \frac{d\bar{v}}{dt} = \frac{dv}{dt} \cdot \frac{t_{ref}}{v_{ref}} \quad (21)$$

Breaking equation (11) into two parts:

$$\dot{\bar{v}} = \frac{t_{ref}}{v_{ref}} \left( -g \cdot \sin(\gamma) \right) + \frac{t_{ref}}{v_{ref}} \cdot A_s \quad (22)$$

$$\text{where } A_s = \frac{T \cdot \cos(\alpha) - D}{m}$$

The term  $A_s$  is the acceleration due to the system forces along the tangent of the orbit.

Substituting in for the gravitational acceleration,  $g$ :

$$\dot{\bar{v}} = \frac{-t_{ref}}{v_{ref}} \cdot \frac{\mu}{r_{ref}^2 \bar{r}^2} \cdot \sin(\gamma) + \frac{t_{ref}}{v_{ref}} \cdot A_s \quad (23)$$

Realizing that  $\bar{g} = \frac{1}{\bar{r}^2}$ :

$$\dot{\bar{v}} = -\bar{g} \cdot \sin(\gamma) + a_s \quad (24)$$

$$a_s = \frac{t_{ref}}{v_{ref}} A_s \quad (25)$$

$$a_s = \frac{t_{ref}}{v_{ref} m_{ref}} \cdot \frac{T \cdot \cos(\alpha) - D}{\bar{m}} \quad (26)$$

Equation (26) would be sufficient in normalizing the velocity differential with thrust and drag expressed as values normalized by the centrifugal force.

$$\bar{T} = \frac{T}{\left( \frac{m_{ref} v_{ref}^2}{r_{ref}} \right)} \quad \text{and} \quad \bar{D} = \frac{D}{\left( \frac{m_{ref} v_{ref}^2}{r_{ref}} \right)} \quad (27)$$

However this form would make the relationship between thrust and drag difficult to visualize. Instead, thrust and drag are divided by the value of drag that the vehicle would experience at the reference altitude. When the vehicle is at the reference altitude the value of the normalized drag would be one while the value of thrust would be in terms of the drag force. A normalized thrust value of 10 would allow the spacecraft to have a thrusting force ten times the force of drag.

$$\bar{T} = \frac{T}{D_{ref}} \quad \text{and} \quad \bar{D} = \frac{D}{D_{ref}} \quad (28)$$

The value for  $D_{ref}$  is defined by the formula for atmospheric drag.

$$D_{ref} = \frac{1}{2} \rho_{ref} v^2 C_D A \quad (29)$$

The variable  $\rho_{ref}$  is the density at the reference altitude,  $C_D$  is the drag Coefficient, and  $A$  is the cross sectional area of the vehicle along the velocity vector. Using this method of referencing thrust and drag equation (26) becomes:

$$a_s = \frac{t_{ref}}{v_{ref} m_{ref}} \cdot D_{ref} \cdot \frac{\bar{T} \cdot \cos(\alpha) - \bar{D}}{\bar{m}} \quad (30)$$

Looking at only the coefficient of the above equation:

$$\frac{t_{ref} D_{ref}}{v_{ref} m_{ref}} = \frac{r_{ref} D_{ref}}{v_{ref}^2 m_{ref}} = \left( \frac{r_{ref} \rho_{ref}}{2} \right) \cdot \left( \frac{C_d A}{m} \right) \quad (31)$$

A final variable is introduced, the ballistic coefficient of the spacecraft,  $B$ . The ballistic coefficient is a function of the vehicle's mass and cross sectional area and is represented by the following equation:

$$B = \frac{m}{C_d A} \quad (32)$$

To normalize the ballistic coefficient an arbitrary reference coefficient was chosen and is shown below:

$$\bar{B} = \frac{B}{\left( \frac{r_{ref} \rho_{ref}}{2} \right)} \quad (33)$$

Using equations (31) through (33) the following can be shown:

$$\frac{t_{ref} D_{ref}}{v_{ref} m_{ref}} = \frac{1}{\bar{B}} \quad (34)$$

Finally, the full non-dimensional form of the velocity differential is determined by combining equation (24) with the above expression.

$$\dot{\bar{v}} = -\bar{g} \cdot \sin(\gamma) + \frac{\bar{T} \cdot \cos(\alpha) - \bar{D}}{\bar{m} \bar{B}} \quad (35)$$

At this point all of the variables have been normalized and the remainder of the non-dimensionalized equations of motion are listed below.

$$\dot{\bar{r}} = \bar{v} \cdot \sin(\gamma) \quad (20)$$

$$\dot{\bar{v}} = -\bar{g} \cdot \sin(\gamma) + \frac{\bar{T} \cdot \cos(\alpha) - \bar{D}}{\bar{m}\bar{B}} \quad (36)$$

$$\dot{\gamma} = \left( \frac{\bar{v}^2}{\bar{r}} - \bar{g} \right) \cdot \frac{\cos(\gamma)}{\bar{v}} + \frac{\bar{T} \cdot \sin(\alpha)}{\bar{m} \cdot \bar{v} \cdot \bar{B}} \quad (37)$$

$$\dot{\bar{m}} = \frac{-\bar{T}}{\bar{v} \cdot \bar{B}} \quad (38)$$

$$\dot{\bar{\theta}} = \frac{\bar{v}}{\bar{r}} \cdot \cos(\gamma) \quad (39)$$



### III. ORBIT RAISING MANEUVERS

#### A. THE LOW EARTH ATMOSPHERE

While drag is considered mainly for air-breathing systems, it is the principal non-gravitational force that affects all objects in low-Earth-orbit. While the density of the atmosphere at orbital altitudes is about  $10^{-11}$  times smaller than at the surface of the Earth, orbital velocities are approximately 7700 m/s. Recalling equation (29) for  $D_{ref}$ , the magnitude of the drag force is proportional to the square of the velocity.

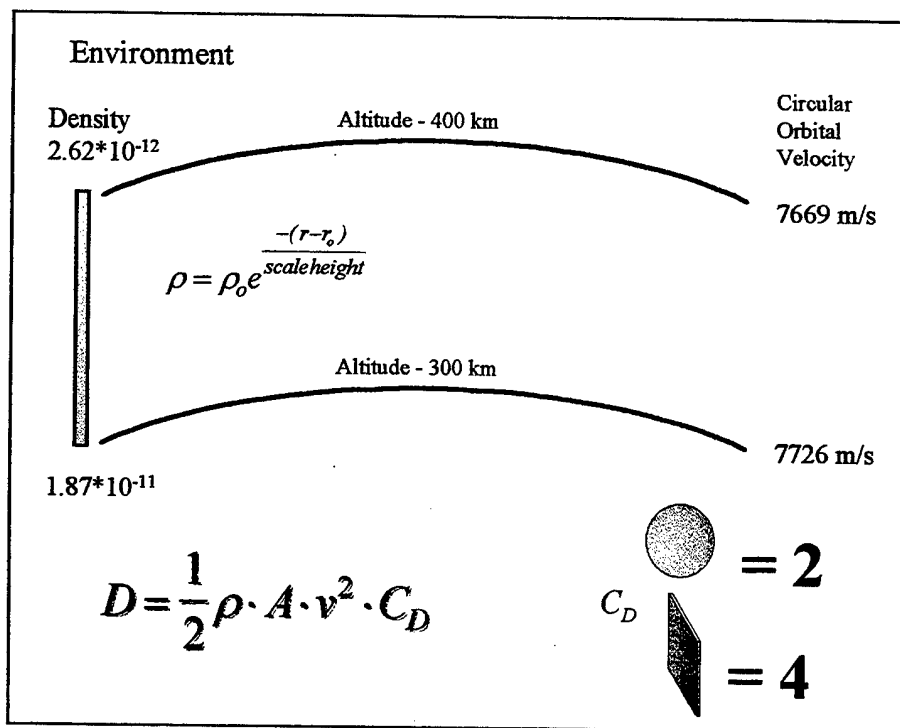


Figure 3-1

Figure 3-1 shows the various inputs into the drag equation for the some typical low-Earth orbit altitudes. The variables combine to make drag a small force that continually acts to perturb the original orbit. By dissipating energy, drag causes the orbit to shrink. As the orbit altitude decreases atmospheric density increases. This increases the net drag force, which increases the rate of orbital energy dissipation. In the determination of orbital motion that includes drag, it is important to model the force of drag upon the vehicle at different altitudes. To accurately describe drag, one must choose a fairly

accurate density model. The Earth's thermosphere, which begins above 90 km in altitude, is the region of the atmosphere that absorbs extreme ultraviolet radiation. The temperatures of the molecules within this region vary widely between day and night due to different levels of UV absorption. Increases in temperature create an increase in atmospheric density. Other disturbances such as geomagnetic activity and solar cycle can vary density values on an hourly basis. Complex models have been used to describe the variations in density, the two most popular being the Jacchia and the Mass Spectrometer Incoherent Scatter (MSIS) models. Numerically, both of the models are complex and approximate atmospheric densities as a function of time and position. To avoid the complexity during implementation of the optimal periodic control problem, a simpler density model was chosen. The main reason for stripping away higher-order terms and variables in the atmospheric density model is to study the problem at the fundamental level. The easiest choice for an atmospheric model would be to assume a constant density model. When looking at orbit reboost problems the difference between upper and lower orbital altitudes is usually fairly small on the order of 10-50 km. The density change between 10 km is less than 20 percent and provides a first guess into the interaction of drag and optimal thrust control. While this difference may seem significant, larger variations occur at the transition between night and day at orbital altitudes.

The next step in adding complexity is to assume an exponential function of altitude for the densities. The advantage of this model is that density closely follows the nominal measured results. The equation for the exponential density model is shown below.

$$\rho = \rho_0 e^{\frac{-(r-r_0)}{h}} \quad (40)$$

The atmosphere is broken into separate bands with reference values given at a specific value taken from the MSIS model. For example, the density at 300 km is  $1.87e-11 \text{ kg/m}^3$  with the scale height,  $h$ , having a value 50.3 km. All altitudes between 275 km to 325 km are referenced to the values at  $r = 300\text{km}$ . A simple MATLAB function called *density*, included in Appendix 1, shows the atmospheric density determined as a function of orbital altitude. Figure 3.2 shows the output of the function (solid line) with the 'X's representing the MSIS values at the given orbital altitude. The exponential model was used in the formulation of the optimal periodic control problem. The function *density* is

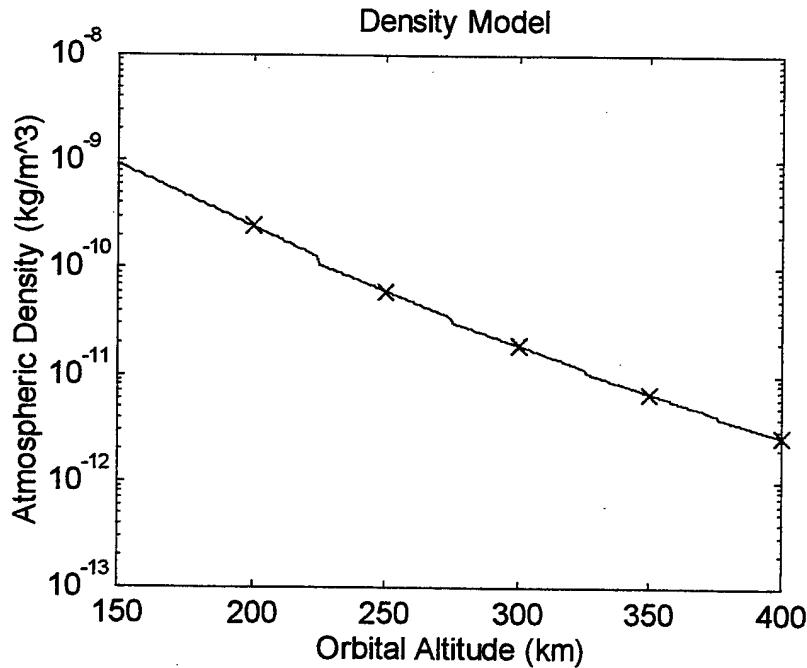


Figure 3-2

called during the numerical iterations to accurately define the atmospheric density at the local altitude. The exponential model, while increasing the complexity of the dynamical model, is a fairly good approximation of the density as a function of altitude.

## B. ORBIT-RAISING MANEUVERS

With the environment and physical laws that govern the motion of the spacecraft, the next step in understanding the orbit maintenance problem is to discuss orbit-raising techniques. Simply put, orbit maintenance is the process that maintains the satellite in a specific region of space during the lifetime of the satellite. As stated earlier, drag is the largest non-gravitational force that affects a spacecraft, specifically in terms of its orbital altitude. The lower altitude limit represents the lowest altitude to which an orbit can decay. Any further loss of altitude would signify a larger force of drag, jeopardizing the continued motion around the Earth. The upper altitude limit is the maximum altitude at which the spacecraft can operate effectively and safely. Reasons for this limit may include the maintaining of earth observation resolution or reduce the risk of high altitude radiation. Obviously of the two limits, the lower limit is the most critical due to the

immediate threat to the spacecraft's lifetime. In formulating the optimal control problem two methods were examined to use the limits as constraints. The first type is the upper bound unconstrained problem. Here the satellite begins at the lower most limit and forced to remain above the minimum altitude. The second type includes an upper bound on the altitude. The difference between the two altitudes is called the orbital band.

Figure 3-3 shows the different types of orbital transfers. The high-energy transfer is

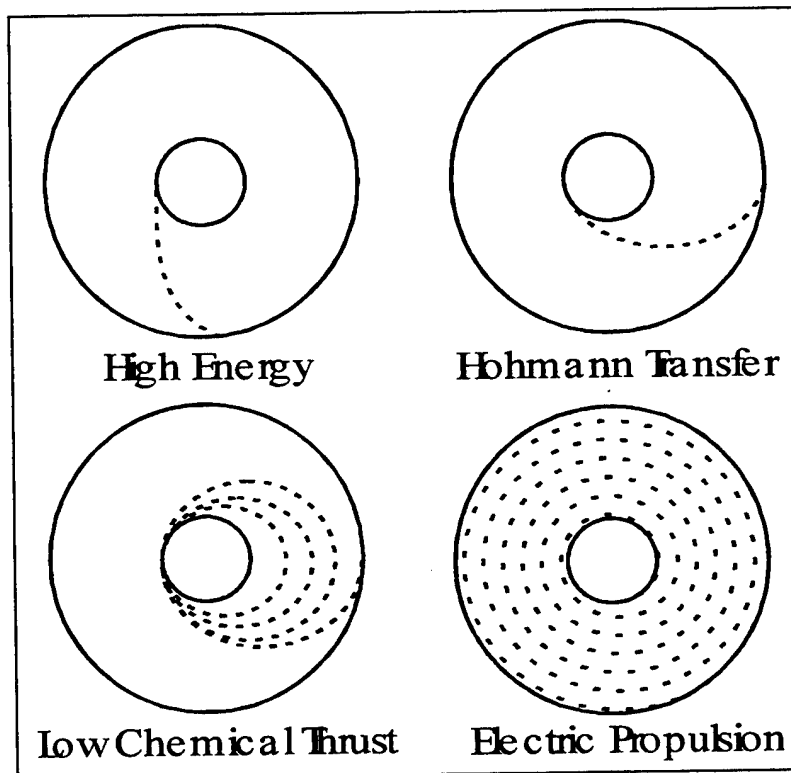
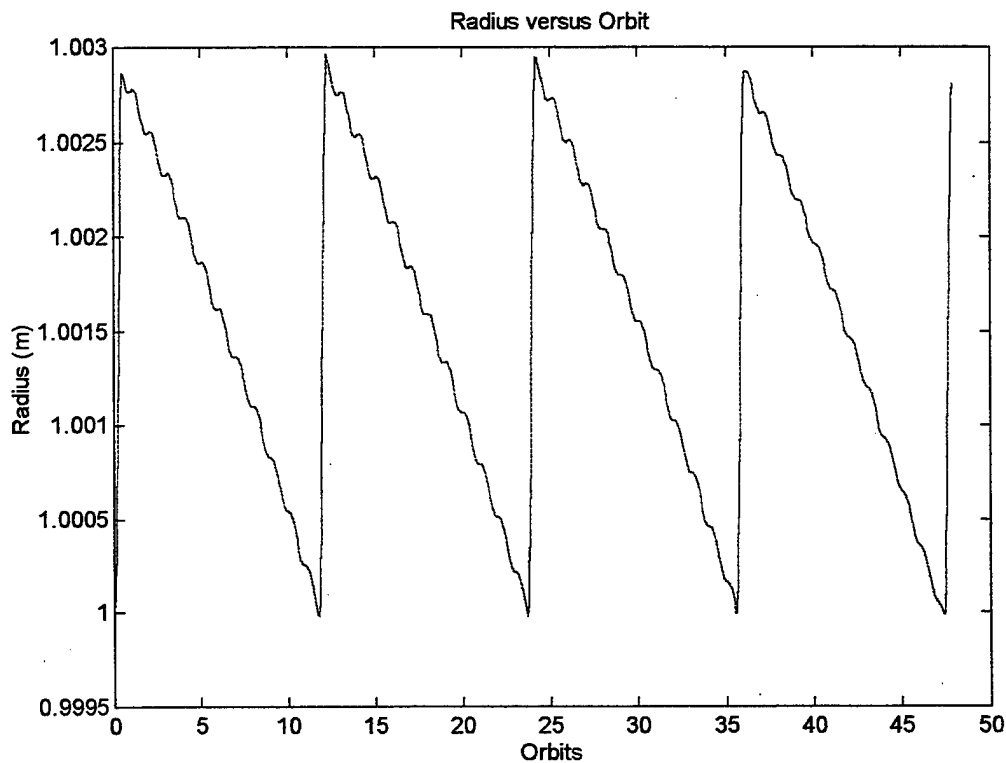


Figure 3-3

the minimum time transfer but uses large amounts of propellant. The second is the most common method of orbit altitude maintenance and is called the Hohmann maneuver, designed to keep the spacecraft within the orbital band. As the vehicle approaches the lower altitude due to drag-induced decay, the first of two thruster burns is applied to the spacecraft to reboost it to the top of its band. After the first burn the vehicle travels approximately along an elliptical transfer orbit which has an apogee equal to the upper altitude limit. As it approaches apogee a second burn is made circularizing the orbit at the upper altitude. After the satellite's orbit is decayed by drag the process is repeated. The

amount of fuel required to accomplish these burns over time is called the propellant budget and directly determines the length of time a satellite will be able to maintain operational status. Figure 3-4 shows the typical Hohmann trajectory. The path was determined by a first order differential solver included in Appendix B called *orbprop*. This program takes the equations of motion with an initial condition and propagates the equations over a fixed period of time. During the orbital propagation an exponential



*Figure 3-4*

density model is used. The actual thrusting logic sequence comes from a separate program called *hoh* that is included in Appendix C. All of the Hohmann thruster burns are impulsive maneuvers, meaning that the burns occur instantaneously with infinite force. The ideal Hohmann neglects gravity and drag loss terms. For real-world applications, a margin must be added to the required propellant mass to account for these losses. Depending on the size of the orbital reboost thruster, between five to ten percent additional propellant may be added to the mass budget.

The final two transfers of Figure 3-3 are also important to mention. First the chemical transfer is a real world application of the Hohmann transfer. Since impulsive burns are infeasible, the transfer burns are broken into distinct parts. As the satellite approaches perigee a finite burn is accomplished, placing the satellite into an elliptical transfer orbit. The apogee of this burn is less than the desired final altitude. The satellite then makes an additional burn at the following perigee, placing it into another elliptical orbit with a higher eccentricity. The process is continued until the apogee height of the transfer orbit is equal to the desired altitude. The total amount of burns nearly approximates the  $\Delta v$  required by the Hohmann transfers. The final burn from Figure 3-3 is the low thrust electrical propulsion transfer. Here the thrust of the electric engine is low and therefore causes a trajectory that spirals from the lower altitude until the higher altitude is reached. This trajectory resembles the opposite of the drag induced decay spiral.

The next maneuver to analyze is called forced Keplerian trajectories (FKT). An FKT is simply a thrust drag cancellation process. The thrusters are fired in opposition of the drag force. This requires that the thruster act in a continuous mode with the ability to change its thrusting force depending on the periodic variations of the local atmospheric density or velocity and altitude for an elliptical orbit. The trajectory would eliminate any altitude loss due to drag. If the orbit were circular the resultant trajectory of an FKT maneuver would remain circular at the original altitude. Comparing this trajectory and propellant requirement versus a Hohmann transfer requires an additional distinction. Since the satellite in a Hohmann maneuver travels from the top of the orbital band to the bottom one must account for the variations of the drag force. Thus to cover the entire orbital band, three different FKT maneuvers were considered. The first, labeled the low-FKT, maintains the satellite at the bottom of the band. The mid-FKT maintains an altitude at the center of the band while a high-FKT keeps the satellite at the top of the band. The same propagation code, *orbprop*, tabulated the propellant used to counteract drag at the three different altitudes. These values were then compared to the fuel requirements of the Hohmann maneuver and shown in Figure 3-5.

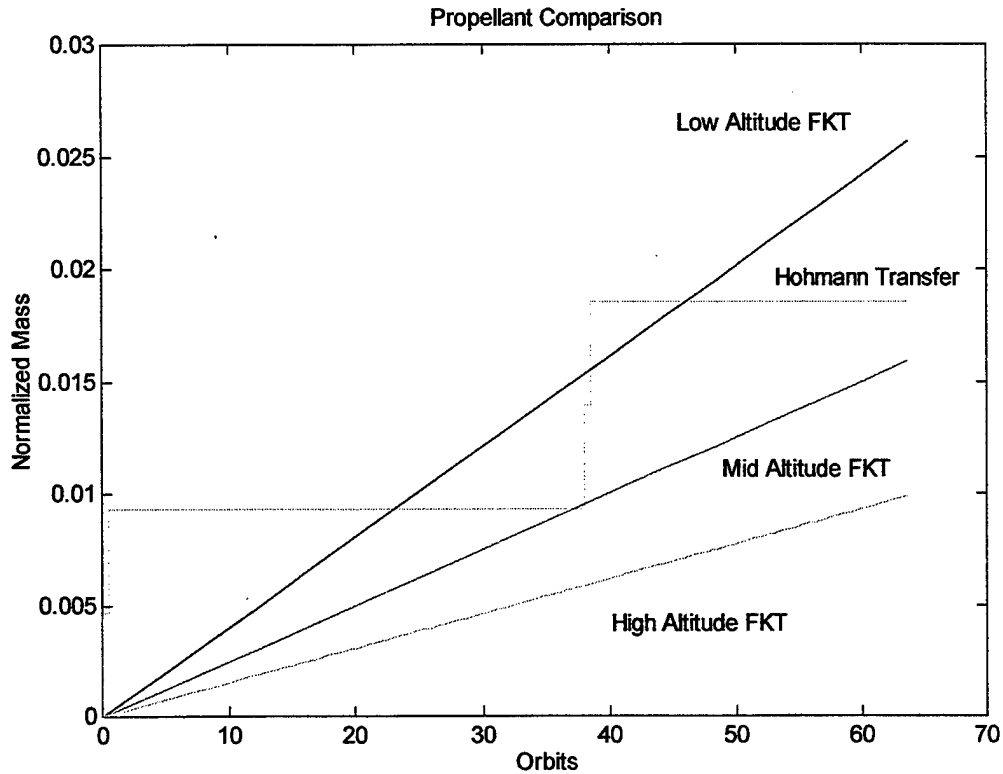


Figure 3-5

The ideal Hohmann is closely approximated to the mid-FKT. To keep a vehicle fixed at the lower altitude limit requires the most fuel of the four cases. The force of drag is the greatest while at the lower altitude. The orbital velocity is higher and the atmospheric density is also higher. Thus the necessary propellant to counteract the higher force must be higher than altitudes where drag is less. A common sense rule of thumb leads one to suggest that to decrease the amount of fuel for orbital maintenance, the satellite must be maintained at a higher altitude. Unfortunately, this reasoning may not be amenable with the vehicle's mission and requirements.

If there are no state constraints, Ross, *et. al* [Ref. 3,4], have shown that the FKT is not the fuel optimal solution. This was accomplished by considering the totality of extremal arcs. Ignoring the special case when the maximum available thrust equals drag, the FKT is not a singular arc and thus not fuel optimal [Ref. 3,4]. Since the Hohmann transfer does not do better than the mid-FKT the periodic Hohmann reboost cannot be the fuel optimal solution as well. Thus there must exist some other trajectory that is more fuel-efficient. If

one considers the drag loss, burn times and orbital positioning some savings of fuel can be made. To determine this fuel-efficient trajectory optimal periodic control theory is explored.



## IV. OPTIMAL PERIODIC CONTROL

### A. PROBLEM FORMULATION

Orbital motion in its simplest form is a cyclic process. Any orbital body subject to no perturbations will travel along a trajectory in a periodic manner with the orbit defined by a specific set of states and period. Periodicity is a further specialization of optimal control theory where the states and controls are cyclic and their values repeat over an optimal period. However before the periodic nature of the theory is examined, the general formulation of the optimal control problem must be established. In differential form of optimal control, the following represents the state and control equations.

$$\dot{x}(t) = f(x(t), u(t), t) \quad t \in \mathcal{R}^+ = [0, \infty) \quad (41)$$
$$u(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_p(t) \end{bmatrix}, \quad x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_m(t) \end{bmatrix}$$

Here the normalized states are represented by the equations (20) and (36)-(39). The controls considered in this problem are thrust,  $T$ , and the thruster angle,  $\alpha$ . This representation develops a simple mathematical form that adequately predicts the response of any system. The history of control inputs from  $[t_0, t_f]$  is called the control history,  $u(\cdot)$ , while the state values over the same time interval is the system's trajectory. The trajectory and the control history must satisfy all of the system state and control constraints in order to be classified as admissible. The constraints reduce the range of values that can be assumed by the state and control variables. In the case of orbital motion several obvious constraints exist. The state equations are constrained in that the mass can never be negative or using a lower altitude limit places an inequality constraint upon the radius. The controls are constrained as well: Thrust is bounded below by zero, the force when not firing, and bounded above by a maximum, derived by the physical limitations of the motor.

The next characteristic of the optimal control problem is the need for a performance measure. Optimal control is defined as the process which minimizes a given performance criterion. In general the performance measure can be written as the following:

$$J = h(x(t_f), t_f) + \int_{t_0}^{t_f} g(u(t), x(t), t) dt \quad (42)$$

For orbit maintenance the main performance measure is to minimize the amount of propellant expended. For periodic control the performance measure becomes an average cost measure. Under the framework of optimal periodic control theory and the specific problem, the performance criteria, or cost function, is simply the average amount of propellant required over an optimal period,  $\tau$ . The  $h$  portion of equation (42), can be eliminated through the use of judicious referencing of the state values. The periodic performance equation becomes:

$$J = \frac{1}{\tau} \int_0^{\tau} g(u(t), x(t), t) dt, \quad (43)$$

The functional  $g$  is dependent on functions containing the controls and the states. To create a fuel-optimal orbit maintenance profile, one examines the minimization of the propellant required over a given period. The representation of the cost function begins with the following equation.

$$J = \frac{m(0) - m(\tau)}{\tau} \quad (44)$$

Neglecting any pressure differences in the nozzle exhaust, an equation for the average thrust becomes:

$$T = \dot{m} \cdot v_e = (m(0) - m(\tau)) \cdot \tau \cdot v_e \quad (45)$$

Putting the cost function in terms of an integral as in equation (43):

$$J_p = \frac{1}{\tau} \int_0^{\tau} \frac{T}{v_e} dt \quad (46)$$

Substituting in the non-dimensional variables the periodic cost function becomes:

$$J_p = \frac{1}{\tau} \int_0^{\tau} \frac{\bar{T}}{v_e \bar{B}} dt \quad (47)$$

The added specialization of optimal control theory is the periodic behavior of the states and controls. Called optimal periodic control (OPC), the significance of this theory lies in its boundary conditions. In standard OPC theory, all the states are periodic. This results in periodic costates as well as a transversality condition, in terms of the

Hamiltonian, from which the optimal period may be determined. One of the goals of periodic control is to identify the optimal period in order to gain the most efficient system. However in the case of orbital motion and maintenance, the problem formulation requires that some of the states are not periodic. For the orbit problem, the radius, velocity, and flight path angle ( $\gamma$ ) are all periodic values while mass and the position angle theta are aperiodic. For instance mass is a state that begins at a certain value and decreases until it is zero. Angular position,  $\theta$ , is also not necessarily periodic. Since some of the values are aperiodic while others are periodic it is difficult to apply the full optimal periodic control theory. Instead the problem is solved by a numerical method that is discussed further in a following section. An advantage of using this method is that the initial conditions along with the fuel-optimal trajectory are obtained. The nonlinear code is given a set of initial guesses for the states and controls and then is free to change the value of the variables assuming they are not manually fixed as a constraint. The initial conditions determined by the optimal code are those which minimize the cost function.

The period, represented by  $\tau$ , can either be determined by the optimal code as a free variable or fixed.

$$\begin{aligned}
 \bar{r}(0) &= \bar{r}(\tau) & \bar{\gamma}(0) &= \bar{\gamma}(\tau) & \bar{v}(0) &= \bar{v}(\tau) \\
 \bar{m}(0) &= 1 & \bar{\theta}(0) &= 0 \\
 \bar{m}(\tau) &= \text{free} & \bar{\theta}(\tau) &= \text{free}
 \end{aligned} \tag{48}$$

Equation (48) shows the boundary conditions for the case where the initial radius, velocity and flight path angle are all free. Variations of the boundary condition set will be explained with each individual case.

A further modification was made to the cost function in order to help quantify any efficiency gained by the numerical code. If one assumes a low FKT trajectory, specifically a satellite that uses a thrust drag cancellation profile at the altitude of its lower limit, the cost function for this trajectory is written below.

$$J_{FKT} = \frac{1}{\tau} \int_0^{\tau} \frac{\bar{T}}{v_e \bar{B}} dt \tag{49}$$

Since the trajectory is an FKT, thrust is equal to drag. Recalling equation (28), thrust is referenced in terms of drag at a specific reference altitude.

$$\bar{T} = \frac{T}{D_{ref}} \quad \text{and} \quad \bar{D} = \frac{D}{D_{ref}} \quad (28)$$

Using this fact, an FKT trajectory would have a value  $\bar{T} = 1$ . Equation (49) becomes:

$$J_{FKT} = \frac{1}{\tau_0} \int \frac{1}{v_e \bar{B}} dt = \frac{1}{v_e \bar{B}} \quad (50)$$

Using equation (50) as a reference one can then divide the cost function obtained in equation (47) to have a referenced performance measure.

$$J = \frac{J_p}{J_{FKT}} = \frac{1}{\tau_0} \int \bar{T} dt \quad (51)$$

This referenced cost function is very relevant when the radius is constrained to never drop below the lower altitude limit. A cost function with a value of one would equal the performance of the low-FKT. A cost function less than one would mean that the optimal trajectory would be more efficient than a low-FKT. In other words, the aim of the code is to determine a  $J$  less than one.

## B. NUMERICAL METHOD

Traditionally, optimal control problems are solved using shooting methods that require the formulation of costate equations obtained from first order necessary optimality conditions. There exist no widely accepted methods and extensive theoretical results for determining the necessary conditions for a partially periodic problem. Therefore the emphasis of this thesis is to solve the optimal periodic control problem directly. A spectral collocation method is used to numerically solve the OPC problem for the proposed states and controls. The spectral collocation method used here has already been successfully applied to solve a class of linear and nonlinear optimal control problems with state and control constraints [Ref. 5,6].

The premise of this method is to discretize the nonlinear control problem at specific points or nodes and convert it into a system of nonlinear algebraic equations with unknown as the values of the states and controls at the nodes. The resulting nonlinear program can be solved by existing routines. In order to create orthogonal polynomial

approximations for the control and state equations a spectral collocation method is employed to determine the values of these functions at the Legendre-Gauss-Lobatto (LGL) points. These points are used as the collocation points and Lagrange polynomials are used as orthogonal trial functions.

Given any function  $F(t)$  that exists over an interval from  $[0, \tau]$  a time transformation must be used to convert it into the interval  $[-1, 1]$ , where the Legendre-Gauss-Lobatto points lie.

$$\begin{aligned} t &= \frac{\tau}{2}(\bar{t} + 1) \\ \text{where} & \\ t \in [0, \tau]; \quad \bar{t} &\in [-1, 1] \end{aligned} \quad (52)$$

Using (43) and (52) the performance function becomes:

$$J = \frac{1}{2} \int_{-1}^1 g(x(\bar{t}), u(\bar{t}), \bar{t}) d\bar{t} \quad (53)$$

And the periodic boundary conditions change to:

$$x(-1) = x(1) \quad (54)$$

Let  $L_N(\bar{t})$  represent the Legendre polynomial of order  $N$ . These polynomials are determined by the following recursive expression:

$$\begin{aligned} (i+1)L_{i+1}(\bar{t}) - (2i+1)\bar{t}L_i(\bar{t}) + iL_{i-1}(\bar{t}) &= 0 \\ \text{where } L_0(\bar{t}) &= 1, L_1(\bar{t}) = \bar{t}, \\ \text{and } i &= 1, 2, \dots, N \end{aligned} \quad (55)$$

Existing numerical codes were used to determine the location of the zeros that give the time interval between the nodes. The  $\bar{t}_i$ , where  $i = 0, 1, \dots, N$ , are defined as  $\bar{t}_0 = -1$ ,  $\bar{t}_N = 1$ , and  $\bar{t}_k$  equal to the zeros of  $\dot{L}_k(\bar{t})$ . The next step is to construct the polynomial approximations by first defining the Lagrange polynomials in terms of  $L_N(\bar{t})$ s.

$$\phi_i(\bar{t}) = \frac{1}{N(N+1)L_N(\bar{t}_i)} \cdot \frac{(\bar{t}^2 - 1)\dot{L}_N(\bar{t})}{\bar{t} - \bar{t}_i} \quad (56)$$

It can be shown that:

$$\phi_i(\bar{t}_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (57)$$

Using the above relations, one can now build the polynomial approximation of the following form:

$$x_i^N(\bar{t}) = \sum_{j=0}^N x_i(\bar{t}_j) \phi_j(\bar{t}), \quad i = 1, 2, \dots, m \quad (58)$$

$$u_i^N(\bar{t}) = \sum_{j=0}^N x_i(\bar{t}_j) \phi_j(\bar{t}), \quad i = 1, 2, \dots, p \quad (59)$$

By differentiating equation (58) and introducing the differentiation matrix  $D_{jk}$  we get the following expression:

$$\dot{x}_i^N(\bar{t}_j) = \sum_{k=0}^N D_{jk} x_i(\bar{t}_k) \quad (60)$$

The differentiation matrix  $D_{jk}$  is defined as follows:

$$D_{jk} = \begin{cases} \frac{L_N(\bar{t}_j)}{L_N(\bar{t}_k)} \cdot \frac{1}{\bar{t}_j - \bar{t}_k} & j \neq k \\ -\frac{N(N+1)}{4} & j = k = 0 \\ \frac{N(N+1)}{4} & j = k = N \\ 0 & \text{otherwise} \end{cases} \quad (61)$$

Placing the equations (59) and (60) in vector form:

$$\begin{aligned} x^N(\bar{t}) &= \sum_{k=0}^N a_k \phi_k(\bar{t}) \\ u^N(\bar{t}) &= \sum_{k=0}^N b_k \phi_k(\bar{t}) \\ \dot{x}^N(\bar{t}) &= \sum_{k=0}^N D_{jk} a_k \end{aligned} \quad (62)$$

where

$$\begin{aligned} a_k &= [a_{1k}, a_{2k}, \dots, a_{mk}] \\ b_k &= [b_{1k}, b_{2k}, \dots, b_{pk}] \end{aligned}$$

The coefficients  $a_k$  and  $b_k$  are yet to be determined but it should be noted that:

$$\begin{aligned} a_k &= x^N(\bar{t}_k) \\ b_k &= u^N(\bar{t}_k) \end{aligned} \quad (63)$$

The next step in the approximation process is to discretize the cost function integral. It can be shown that the cost function can be rewritten in the following form.

$$J^N = \frac{1}{2} \sum_{k=0}^N g(a_k, b_k, \bar{t}_k) w_k \quad (64)$$

Where the term  $w_k$  are the weights given at each node and are defined as:

$$w_k = \frac{2}{N(N+1)} \cdot \frac{1}{[L_N(\bar{t}_k)]^2} \quad (65)$$

The state equations are discretized by substitution and collocation at the LGL nodes,  $\bar{t}_k$ .

The equations are then converted into the following series of algebraic equations in terms of the vectors  $a_j$  and  $b_j$ .

$$A_k = \frac{\tau}{2} f(x^N(\bar{t}_k), u^N(\bar{t}_k)) - d_k = 0,$$

$$\text{where } d_k = \sum_{j=0}^N D_{kj} a_j \quad (66)$$

$$\text{and } k = 0, \dots, N$$

Similarly the system constraints can be approximated in the same manner.

$$g(x(\bar{t}), u(\bar{t})) \leq 0 \quad (67)$$

$$B_k = g(x^N(\bar{t}_k), u^N(\bar{t}_k)) \leq 0,$$

### C. NONLINEAR PROGRAMMING CODE

The next section will describe the various computer codes that were built in order to solve the optimal control problem directly. Each program represents a single or series of functions. When used in conjunction with each other the result is a series of commands which solves the optimal periodic control problem, checks the values against a differential solver, compares the results against different trajectories and finally records the data. All the codes are programmed in MATLAB and it is assumed that the reader has a background in the MATLAB language.

## 1. *Orbprop* – Appendix B

Mentioned in the previous section, *orbprop* is a first order differential equation solver which propagates the equations of motion given a set of initial conditions. Figure 4.1 shows the declarations for the five states.

```
% States
%   x(1) = radius
%   x(2) = velocity
%   x(3) = angle gamma
%   x(4) = mass
%   x(5) = angle theta
```

*Figure 4-1*

*Orbprop* is a function that is called by other functions. The program requires that initial conditions be given for each of the states and time. The program then takes a profile for the controls, thrust and thruster angle, and chooses time steps in order to begin propagating the values. The profile for the controls depends on which program calls *orbprop* into action. For instance, the results from the optimal code can be transferred into a time control history that this program can use to propagate the states. Other cases include setting the thrust, denoted by  $T_p$  in Figure 4-2, to equal drag for the FKT trajectory while another trajectory would call for thrust to equal zero allowing orbital decay in a “free fall” manner. Figure 4-2 is an excerpt from the program that shows the differential equations of motion.  $T_p$  and  $\alpha$  represent the interpolated values of the thrust and thruster angle respectively. Since the propagator uses different time steps than the location of the LGL points, interpolation is required to define the controls at these new points.



```

As = (Tp*cos(alphap)-D)/(x(4)*B);
An = Tp*sin(alphap)/(x(4)*B);

ddot(1) = x(2)*sin(x(3));
ddot(2) = -g*sin(x(3))+As;
ddot(3) = (x(2)^2/x(1)-g)*(cos(x(3))/x(2))+An/x(2);
ddot(4) = -abs(Tp)/(ve*B);
ddot(5) = x(2)*(cos(x(3))/x(1));

```

Figure 4-2

The expressions for  $A_s$  and  $A_n$  in the figure represent the only terms in which the controls affect the equations of motion. The five “ddot” expressions are the normalized equations of motion derived earlier (equations (20),(36)-(39)).

## 2. *Orbopt* – Appendix D

This program is the macro program that controls the flow of the optimal periodic control problem. It begins with a series of variable declarations that give the problem physical meaning and definition. The next step calculates the normalized ballistic coefficient. This term is the only term that relates to the actual spacecraft being modeled. Its mass, area, and coefficient of drag are all contained within this single term. A very large value for the ballistic coefficient means that the spacecraft is either very massive and thus not greatly affected by drag or that the spacecraft is small and drag resistant. Low values of  $\bar{B}$  would suggest that the spacecraft is very susceptible to drag due to its small mass and/or large surface area.

The next section of this program initializes the vector that contains all of the states and controls, named “aop”. Figure 4-3 calls a MATLAB program included in the Optimization Toolbox called *constr*. This function finds the constrained minimum of a function of several variables.

```

aop=constr('orbcrit',aop,options,vlb,vub);

```

Figure 4-3

This command calls the function file called *orbcrit*, which is discussed in the following section. The options are a set of choices the user can alter to customize the minimization process including setting the minimization tolerances, number of iterations, etc. The “vlb” and “vub” are the optional lower and upper bounds. In most cases these were not used, instead all altitude constraints were entered in as inequality constraint equations. After the

```
r=aop(1:n);
v = aop(n+1:2*n);
gamma = aop(2*n+1:3*n);
mass = aop(3*n+1:4*n);
T = aop(4*n+1:5*n);
alpha = aop(5*n+1:6*n);
theta = aop(6*n+1:7*n);
```

*Figure 4-4*

successful completion of the optimization routine the vector “aop” was defined into all of the states and controls shown in Figure 4-4. During this final declaration the variables represent the optimal states and controls which minimize the given cost function. These variables will later be compared to different trajectories in order to measure the effectiveness of the minimization code.

### 3. *Orbcrit* – Appendix E

This program is the center of the spectral collocation method. The code begins with the creation of the cost function. Figure 4-5 contains a section of *orbcrit* that

```
for i=1:n
    fn(i)=aop(4*n+i);
end;
costfn= 1/(2)*sum(w.*fn');
```

*Figure 4-5*

pertains to the creation of the cost function. The vector “fn” is given the control history values of the thrust and is then summed in the “costfn” expression. Recalling equation (51), the derived expression for the reference cost function;

$$J = \frac{J_p}{J_{FKT}} = \frac{1}{\tau_0} \int \bar{T} dt \quad (51)$$

The cost function is simply the sum of the weighted values of the thrust.

The next part in the *orbcrit* code is the creation of the state constraint equations.

Figure 4.6 shows the five state constraint equations.

```

% Radius
g(i) = (2/tau)*sum(Dn(i,:).*aop(1:n))-aop(n+i)*sin(aop(2*n+i));
% Velocity
g(i+n) = (2/tau)*sum(Dn(i,:).*aop(n+1:2*n))+(G(i)*sin(aop(2*n+i)))-Ast;
% Gamma
g(i+2*n) = (2/tau)*sum(Dn(i,:).*aop(2*n+1:3*n));
g(i+2*n) = g(i+2*n)-(((aop(n+i)^2)/aop(i))-G(i))*(cos(aop(2*n+i))/aop(i+n));
g(i+2*n) = g(i+2*n)- Ant/aop(n+i);
% Mass
g(i+3*n) = (2/tau)*sum(Dn(i,:).*aop(3*n+1:4*n))+(aop(4*n+i)/(ve*B));
% Theta
g(i+4*n) = (2/tau)*sum(Dn(i,:).*aop(6*n+1:7*n))-aop(n+i)/aop(i)*cos(aop(2*n+i));

```

Figure 4-6

Before discussing Figure 4-6 in detail a brief explanation is required concerning the way in which constraints are implemented for the optimization routine. All constraints are referenced to zero. For example, to place a constraint on the initial radius the first element of the radius vector is modified.

$$r(t_0) = 1 \quad (68)$$

Recalling that the computer code uses the vector “aop” to represent the states and controls and that radius occupies the first  $n$  terms (where  $n$  equals the number of LGL points) of the “aop” matrix, equation (68) can be referenced to zero and put in the correct form.

$$aop(1) - 1 = 0 \quad (69)$$

The process for inequality constraints is similar. In order to place an upper bound on the amount of thrust a given engine can produce, the following expression would be used.

$$T \leq 5 \quad (70)$$

Figure 4-5 shows the part of the “aop” matrix that relates to thrust. Equation (70) becomes:

$$aop(4 * n + i) - 5 \leq 0 \quad \text{where } i = 1..n \quad (71)$$

Equation (71) prevents any terms within the time history of thrust to be above 5 normalized units of thrust.

Figure 4-6 uses the format required by MATLAB and creates 5n equality constraints for the five states. Taking the first equation, the radius state equation, allows closer examination of the constraint construction. Recalling equations (66) and (20):

$$A_k = \frac{\tau}{2} f(x^N(t_k), u^N(t_k)) - d_k = 0,$$

$$\text{where } d_k = \sum_{j=0}^N D_{kj} a_j \quad (66)$$

$$\text{and } k = 0, \dots, N$$

$$\dot{r} = \bar{v} \cdot \sin(\gamma) \quad (20)$$

Substituting equation (20) defined at the individual time steps,  $t_k$ , into equation (66) the constraint equation becomes:

$$A_{1k} = \frac{2}{\tau} \sum_{l=0}^N D_{kl} r(t_l) - v(t_k) \sin(\gamma(t_k)) = 0 \quad (72)$$

From Figure 4-4 the expressions for the states can be substituted in equation (72) to transform the equation in terms of the “aop” vector.

$$A_{1k} = \frac{2}{\tau} \sum_{l=0}^N D_{kl} aop(l) - aop(n+k) \cdot \sin(aop(2n+k)) = 0 \quad (73)$$

The first term,  $(2/\tau) * \text{sum}(Dn(i,:))$ , follows the spectral collection method where “Dn” is the differentiation matrix governed by equation (61) shown below.

$$D_{jk} = \begin{cases} \frac{L_N(t_j)}{L_N(t_k)} \cdot \frac{1}{t_j - t_k} & j \neq k \\ \frac{-N(N+1)}{4} & j = k = 0 \\ \frac{N(N+1)}{4} & j = k = N \\ 0 & \text{otherwise} \end{cases} \quad (61)$$

The values of the radius are continually iterated as the program is called repeatedly by the *constr* MATLAB function. The other four state constraint equations all are the same format, each containing the equation of motion for the particular state.

The next two sections of *orbcrit* define the physical constraints and periodicity of the orbit maintenance problem. The remaining equality constraints are shown in Figure 4-7.

Periodic Constraints		
$g(5*n+1) = aop(1)-aop(n);$	$r$	
$g(5*n+4) = aop(n+1)-aop(2*n);$	$v$	
$g(5*n+2) = aop(2*n+1)-aop(3*n);$	$\gamma$	
Aperiodic Constraints		
$g(5*n+5) = aop(3*n+1)-1;$	$m$	
$g(5*n+3) = aop(6*n+1);$	$\theta$	

Figure 4-7

Only the final two expressions from Figure 4-7 are given fixed values. One of the equality constraints pertains to the initial value of the angle theta, which is set to zero. Mass is initially set to one through the last equality constraint of Figure 4-7. The other three equality expressions allow radius, velocity, and gamma to be periodic variables; the initial value of the state is equal to its final value. First is a series of inequality constraints shown in Figure 4-8.

$g(i+5*n+5) = - aop(4*n+i);$ $g(6*n+5+i) = aop(4*n+i)-5;$
---

Figure 4-8

In the most basic formulation of the problem, the only inequality constraint considered was the upper and lower bound of the thrust. While most modern day thrusters are either full on or full off, thrust was modeled as an engine capable of a full range of throttling values, ranging from off, a value of 0, to full on, in this case a value of 5.

Soon after the code was initially tested a new constraint was added to the list in Figure 4-8. This constraint forced the initial value (and hence its final value) of orbital radius to be a predefined location in space, normally at the reference altitude. The reason for this

constraint can best be explained as follows. The cost function, as noted before, relies on the amount of thrust used over the optimal period. Intuitively, the amount of drag at higher altitudes is less than the force of drag at lower altitudes. In an attempt to lower the cost function the code tries to force the radius, initial and final to be as high as possible. Given enough computing power and an infinite number of minimization iterations the radius should be infinity where the atmospheric density is zero, allowing the spacecraft to orbit the Earth indefinitely at no cost. Thus to study the effectiveness of this code with a physically feasible problem, the radius was set to an initial value. Consequently, since the radius is a periodic variable the final value was also set to equal the initial altitude.

#### **4. Other Programs – Appendices F and Beyond**

A series of additional programs used to study aspects of the fuel-optimal problem are included in the appendices. Several of these programs deal directly with the spectral collocation method. The code *diffm* creates the differentiation matrix required by the collocation method and that is used in the program *orbcrit*. The programs *lobatto*, *mxt*, *mxtj*, and *tqr*, written by Professor Bill Gragg of the Naval Postgraduate School, computes the abscissa and weights for the n-point LGL quadrature problem.

Variations of the *orbcrit* and *orbopt* programs are included. During the course of this thesis, modifications to the constraint set or initial state and control guess created a series of modified programs. The different programs will be discussed in the following sections, which discuss several aspects of the fuel-optimal trajectories.

The final group of programs is the programs that use the *orbprop* code and are used to compare the trajectories derived by the optimization process versus steady state propagated trajectories.

## V. DISCUSSION OF RESULTS

### A. INTRODUCTION

While attempting to find a fuel-optimal orbit maintenance trajectory was the main goal of this research, a secondary goal was to test the implementation of the spectral collocation method through the use of non-linear programming to directly solve the optimal periodic control (OPC) problem. The first step was to build a framework that contained the computer codes required to solve this type of problem in accordance with the OPC theory. The most basic codes were discussed in the previous chapter. To test the series of programs, the most general case of the fuel-optimal problem was attempted. The results were used to identify several trends. Using these conclusions, the general programs were modified by either changing the system constraints, boundary conditions, or reducing the number of free variables. As a result of these modifications, the orbit maintenance problem was studied more realistically by defining physical limits and constraints of current day systems. For example, the codes were modified to include an upper altitude limit. This constraint causes the trajectory to be restricted within an orbital band. Satellites that possess limited communications ability or those that depended on imagery resolution are examples of an altitude-limited platform. Each of the special cases examined are discussed in detail in following sections.

### B. THE FREE CASE

The most general problem formulation allows defining the satellites physical characteristics, an initial altitude, and a set of initial guesses for the state and control histories. The bulk of the satellite's characteristics is included in the equation for the normalized ballistic coefficient.

$$\bar{B} = \frac{B}{\left( \frac{r_{ref} \rho_{ref}}{2} \right)} \quad (33)$$

Recalling equation (32):

$$B = \frac{m}{C_d A} \quad (32)$$

Using equations (32) and (33) the value of the normalized ballistic coefficient is shown below:

$$\bar{B} = \frac{2 \cdot m}{C_d A \cdot r_{ref} \rho_{ref}} \quad (74)$$

The reference altitude and density are predefined values and are mutually dependent. The spacecraft physical properties are included in the following ratio.

$$\text{Physical Characteristics} = \frac{m}{C_d A} \quad (75)$$

In order to highlight the effects of drag a satellite with a low ratio of mass to area is desired. The reason behind this idea is that a difference in propellant usage under varying drag conditions is a small quantity. By increasing the losses due to drag the value of one trajectory can be more easily identified against another. For purposes of this work, a generic satellite was created that leveraged the fact that drag was the major non-conservative perturbing force.

<b>Generic Spacecraft Physical Characteristics</b>	
Spacecraft Area	500 m <sup>2</sup>
Spacecraft Mass	3000 kg
Maximum Thrust	3.5 N ( $\bar{T} = 5.0$ )
Altitude	300 km
Density	1.87*10 <sup>-11</sup> kg/m <sup>3</sup>
Coefficient of Drag	2.35

*Figure 5-1*

These values are for a theoretical spacecraft but are reasonable for certain kind of platforms, such as space-based radars, large antenna platforms, or inflatables. Figure 5-2



shows the normalized ballistic coefficient for several different systems assuming a reference altitude of 300 km.

<b>Ballistic Coefficient</b>	
Generic Spacecraft	$4.09 \cdot 10^4$
ISS – DACT 6	$1.26 \cdot 10^6$
Space Telescope	$4.72 \cdot 10^5$
Landsat-1	$4.04 \cdot 10^5$
Echo-1 (comms)	$8.24 \cdot 10^2$

*Figure 5-2*

The other characteristic of the generic spacecraft is a measure of the effectiveness of its control, or simply the size of its orbital maintenance thruster. Defined in the normalization section, thrust is placed in terms of the force of drag at the reference altitude. Recalling equation (29):

$$D_{ref} = \frac{1}{2} \rho_{ref} v^2 C_D A \quad (29)$$

Using the values for the generic spacecraft,  $D_{ref}$  would equal approximately 0.7 N. Current orbit maintenance thrusters have thrusts ranging from under 1 N to over 400 N.<sup>5</sup> In normalized terms a thrust equal to one would be 0.7 N. The thruster modeled in the generic spacecraft has a maximum of 5 or about 3.5 N available. Other thruster sizes were examined and the results will be discussed in following sections.

For the free case, the thruster was allowed to fire in any direction relative to the instantaneous satellite velocity vector. The only other constraint placed upon the system was that the initial altitude was fixed to the reference altitude of 300 km. Since the spectral collocation method requires the user to input a series of initial guesses, a series of initial values were placed into the “aop” vector. Unfortunately, initial guesses do have some influence on the results obtained from the optimal control problem. Thus after judicious experimentation the initial guesses were made as given in figure 5-3.

Initial guesses for the Optimization Code		
	Physical Value	Normalized Value
Radius	6678 km	1
Velocity	7769 m/s	1
Mass	3000 kg	1
Gamma	1.1 radians	1.1
Thrust	3.5 N	5.0
Tau	720 minutes	50 TU
All Others	0	0

*Figure 5-v-3*

With all the information entered into the code, the optimal control program, *orbopt* is run within MATLAB. The code is run primarily on an Intel Pentium processor operating at 233 MHz. Total run time for this case was approximately 2 hours. The *constr.m* code in MATLAB takes the initial guesses and calculates the given cost function. It then creates a set of gradient information and chooses new values for the controls and states and recalculates the cost function. If the cost function is lower than the previous value, the next iteration occurs. The process continues until all of the state and control tolerances are met with the minimum cost function.

The results are tabulated by a program called *orbres*, included in the appendix, and then plotted by the program *orbconv*. Figure 5-4 shows the resulting plots for 4 of the 5 states versus time, which is proportional to the fifth state the angle theta.

Figure 5-4 shows several points worthy of notice. The spacecraft is boosted to a higher altitude and then allowed to decay to the original altitude. The flight path angle,

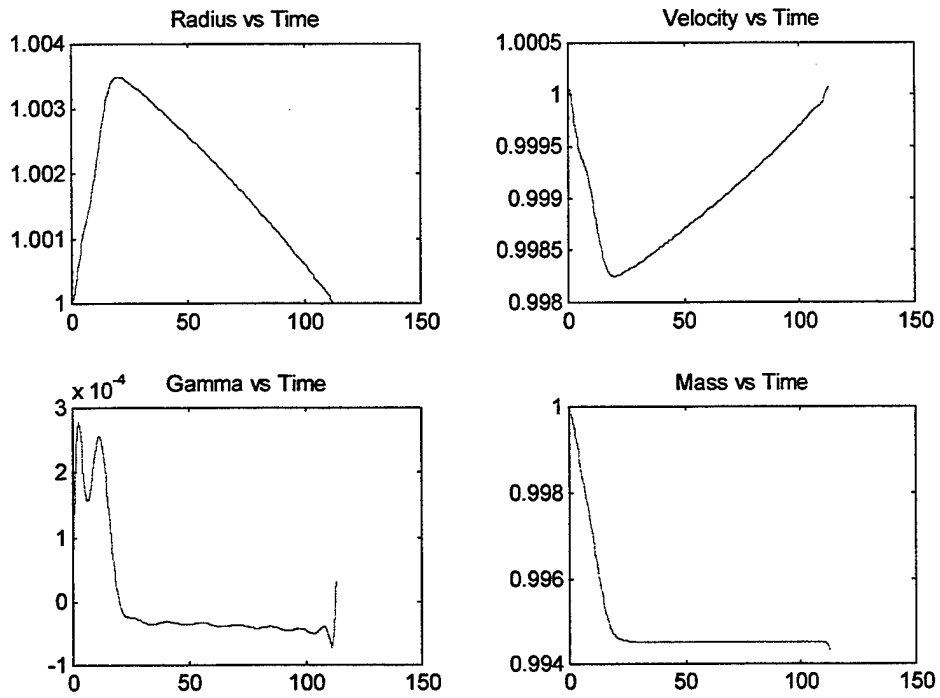


Figure 5-4

gamma, is very small with a maximum of 0.00028 radians (0.016 degrees) indicating that

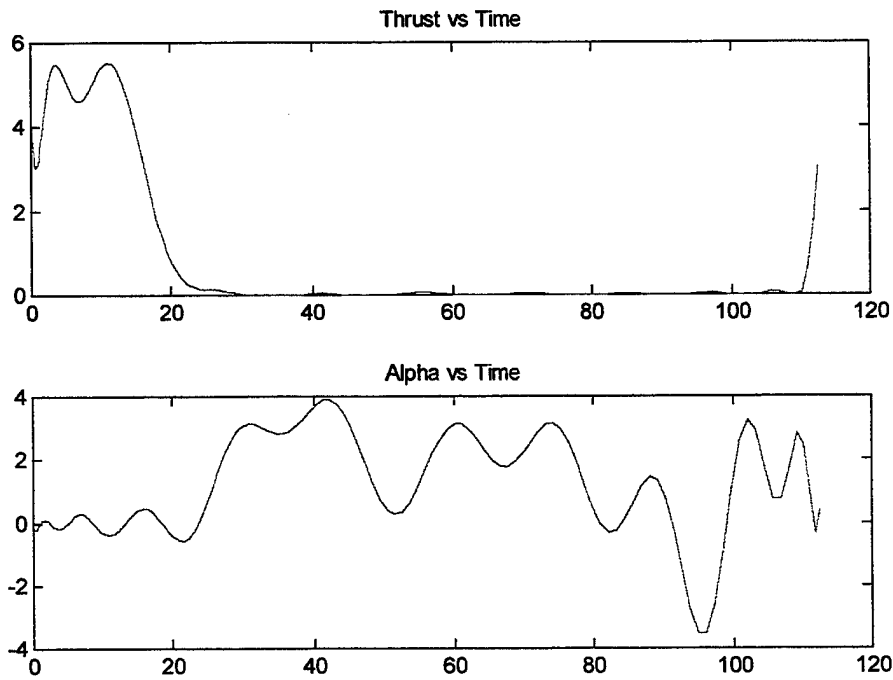


Figure 5-5

the orbit is nearly circular throughout the transfer. Mass is expended rapidly during the first few orbits and then is constant throughout the coast/decay portion. Figure 5-5 shows a plot of the two controls of the spacecraft, the thrust and the thruster angle,  $\alpha$ .

The majority of all thrust is accomplished in the first 3 orbits of the spacecraft (approximately 20 radians) with an additional burn at the end of the period. The important point to notice is that the thrust profile is not a finite-Hohmann transfer burn. The optimal thrust curve dips and peaks meaning that it is not a bang-bang solution. In addition, unlike a Hohmann transfer, the optimal trajectory allows the thruster angle, measured in the variable alpha, to change. The value of alpha range from  $-0.1436$  to  $0.3854$  radians while the thruster is firing. Figure 5-6 shows graphically the variation of alpha during the transfer burn.

This range of thruster cant angles is a normal departure from current orbital transfer

## Variation of Alpha

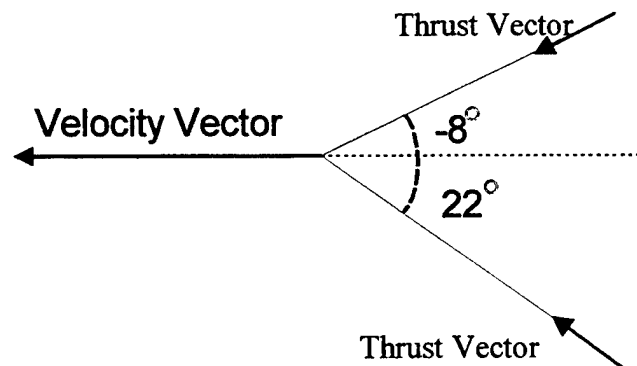


Figure 5-6

maneuvers, which use burns in the direction of the velocity vector only. However when the thruster is off, (i.e.  $T = 0$ ), the thruster angle wanders between its upper and lower limits. This example used  $\pm 180^\circ$  as the bounds for this value. The angle at which the thruster points is only crucial during engine firing. Figure 5-5 shows the wandering of alpha during the decay portion of the simulation. Alpha returns to the neighborhood of zero during the small thruster firing at the end of the run.

The final burn is also an important characteristic of the trajectory. Recall that three of the states; radius, velocity, and gamma, are periodic. The initial values must equal the

final values. As the satellite decays, it approaches the initial altitude and velocity. However its angle gamma, measured between the velocity vector to a line tangential to the radius vector, is negative or tending to point slightly towards the Earth. The final burn is a correction to the angle gamma. The burn rotates the velocity vector to match its original direction.

The next result of this simulation is to examine the cost function. In actuality, the cost function is a ratio between the fuel required for the optimal trajectory and the fuel required for an FKT profile at the starting altitude. Equation (51) shows the ratio mathematically.

$$J = \frac{J_p}{J_{FKT}} = \frac{1}{\tau_0} \int \bar{T} dt \quad (51)$$

Any cost function that is less than one means that the trajectory used less fuel than the FKT trajectory. For the case just considered, the cost function equals 0.784. This means that the optimal trajectory uses 22% less fuel than a trajectory that would follow a thrust-drag cancellation path at the original altitude. Taking a closer look at the physical characteristics of the trajectories helps explain the large difference in fuel. A satellite that is at a higher altitude than another will experience lower drag. The lower the drag, the less the fuel expended to keep the vehicle flying. Density and orbital velocity decrease with increasing orbital altitude. Acknowledging this physical characteristic, a first guess on trajectory design would be to boost the satellite as high as possible and let it decay to its original altitude. The optimal trajectory resembles that concept. Figure 5-3 shows the radius increasing to a maximum after 3 orbits and then decay towards the original altitude. Thus with this in mind one can examine the trajectory more closely and determine during which parts of the flight path might hold savings over traditional orbit maneuvers. The main difference lies in the initial burn and will be explored further in following sections.

The next parameter to examine from the simulation results is the value for the optimal period,  $\tau$ . The value of tau for this example is approximately 110 normalized time units, or just over one day. In a perfect simulation, one would expect that the optimal period would mean that the satellite must follow the trajectory repeatedly for the lowest fuel orbit maintenance consumption. This idea that the period is a definable value is taken from optimal periodic control theory and its application to air-breathing platforms. For

atmospheric flight, periodic controls generate trajectories that are lower in cost than steady state continuous control. It is a reasonable assumption that this would also hold for atmospheric space flight. However there exists a major difference between the two regimes. Thruster performance in low Earth orbit is nearly independent of orbital altitude while air-breathing engine performance relies heavily on atmospheric density, oxygen content and temperature. Drag forces are lowered with higher altitudes in the atmosphere but as altitude increases, engine performance decreases. At orbital altitudes the benefit of higher altitudes and lower drag is not offset by a decrease in engine performance. Thus there is no optimal point in space for the minimum trajectory. This suggests that the optimum altitude to fly would numerically be at infinity.

Examining the optimal tau from this example one asks why is there an optimal period at all. If the optimal orbital altitude were infinity in order to achieve the fuel-optimal trajectory the satellite would need to be boosted to an infinite altitude and allowed to decay for an infinite time. In actuality, the amount that the satellite can be boosted is limited by the amount of propellant mass the satellite carries. The generic spacecraft is a 3000 kg vehicle with perhaps 40 percent propellant mass. With 1200 kg of propellant the spacecraft would optimally be boosted as high as possible. The resulting period for the single burn and subsequent decay would be on the order of years up to infinity. Unfortunately the computer with its associated non-linear programming code has a difficult time modeling infinity. The program runs in an iterative method beginning with the initial guesses and recalculating the cost function after the gradients have been computed. The initial guess for the generic satellite example was 50 time units. The code increased tau to 112 time units. The code increases the period in order to lower the cost function since a longer period allows for a higher boost and subsequent decay.

However to prove that this value is the optimal period for this spacecraft and altitude, the code should be able to reproduce the same states and controls given a different set of initial guesses. For a second run the code was given the same initial conditions as the original example excluding the value of tau which was set at a value of 112 time units, the original example's optimal tau. The simulation took just over 2 hours to run and gave a new optimal tau of 5513 time units, about 56 days. Thus one can conclude that the original example did not give the optimal tau.

The cost function for this second run is 0.1801, or in terms of propellant, 18% of the

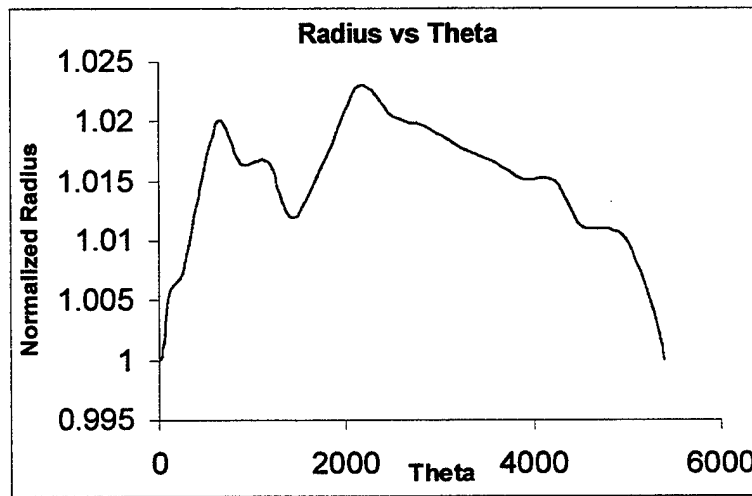
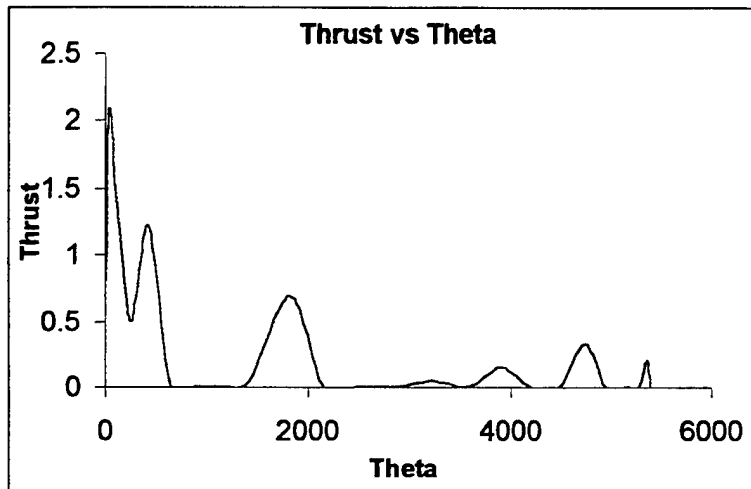


Figure 5-7

fuel required for an FKT trajectory at the original altitude over the determined period. Again the code boosts the spacecraft to a high altitude and then allows a decay to the original height. In the second run the maximum altitude was approximately 1.023 distance units (6831 km) or about 154 km above the original altitude. The first run maximum altitude change was approximately 23 km. The larger altitude change allows a lower cost function over a longer period. Figure 5-7 shows the altitude profile for the second run. One of the problems with the second run is the number of LGL points compared with the length of the simulation. For both runs the minimization program used 24 LGL points. At each of the points the states and controls are known but in the second example only 24 known points are spread out over 5500 time units. The simple correction for this would be to increase the number of nodes. However this method has difficulties with high values

of “n”. The higher complexity and larger matrices have the opposite desired effect. Computing inaccuracies and matrix scaling problems occur. These problems will be discussed more fully in following sections. The lack of detailed representation in the states and controls is one of the reasons the two graphs for the radius versus theta are different between the two runs. Even with the differences, both graphs imply that the trajectory with the minimum cost requires an initial boost to a higher altitude followed by a coast period towards the original altitude. Figure 5-8 shows the thrust versus theta for the

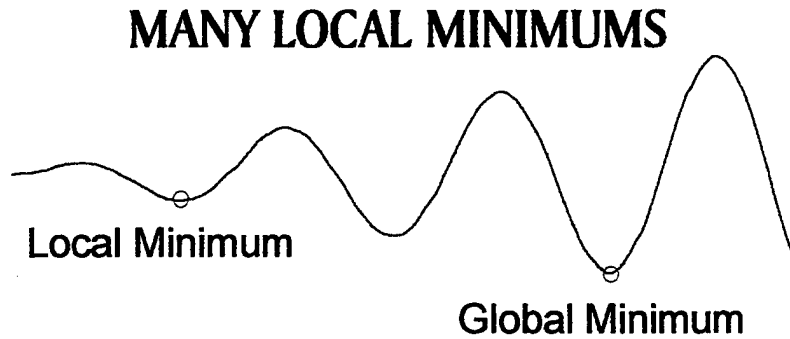


*Figure 5-8*

second run. The two graphs for the thrust are more similar. The initial thrusts are large with two discernable peaks. While the second run has more thruster activity after the initial boost, the thrust is usually small when compared to the initial burns.

Numerically the program iterates upon the initial guess and continues until the constraints on the state and control variables are within given tolerances and when the set of perturbing gradients are below a given value. When guesses are sufficiently away from the optimum values these tolerances may be satisfied at points of local minimums. Figure 5-9 is an attempt to visualize this conclusion and does not represent the actual plot of the system's space.





*Figure 5-9*

Numerically the code approaches the local minimum close to the initial guess. If the gradients are small enough the cost function on either side of the minimum is higher. If the minimum is a local minimum then the code focuses on that location ignoring the possibility of a global minimum location.

If the procedure were to be repeated with the substitution of 5513 time units entered for the initial period the result would be an optimal period of several magnitudes higher. Given enough computing time and power the code would approach infinity as new states and controls are determined to produce a trajectory that would send a spacecraft as high as possible in a single burn to escape the effects of atmospheric drag. The next step in the search for the fuel-optimal trajectories is to fix the orbit maintenance period. When the value of tau is free the code tends to look towards infinity. The reasonable approach is then to look at fixed periods of time to determine if the manner in which burns are accomplished can lead to a one solution that is better than another. This type of analysis can be extended to many different missions that are used today. For example, the International Space Station (ISS) has an orbit maintenance plan that calls for the station to be at a specific altitude in order to rendezvous with the Space Shuttle at specific dates. The ISS planners calculate how high the station is to be reboosted dependant on the time between rendezvous. The next section will examine the optimal trajectories for a given fixed orbit maintenance period.

### C. FIXED TAU

This section takes a look at the results of the optimization code when a fixed value for the orbital maintenance period is used. In actuality, this section will deal with the numerical accuracy and effectiveness of minimization by the non-linear programming code. Recall that the second run of the previous section had a very low cost function but the plot for the radius did not have a smooth first burn but allowed numerous interior burns and decays. The following table shows the masses and periods of the first two runs.

	<i>Check</i> Program	First Run	Second Run
Period	3014.3	112.6	5513.4
Propellant Used	0.0311	0.0057	0.0637

Figure 5-10

The column labeled “*Check program*” is another code that estimates the fuel required for a continuous burn to the desired altitude followed by the time to decay to the original altitude. The numbers shown in this column in figure 5-10 reflect a burn to the same maximum altitude of the second run. This program will be further explained in the following sections where the optimality of the trajectories is discussed. Using the values in figure 5-10, a propellant consumption versus time plot is included in figure 5-11. The periods for the three different trajectory calculations are different; therefore, by extending the overall comparison period by extrapolation, a clearer picture of which trajectory is the most efficient is obtained.

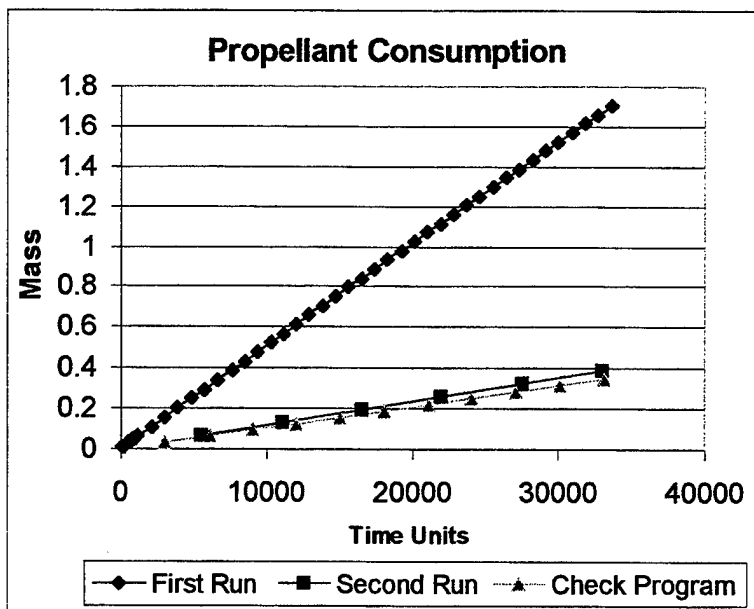


Figure 5-11

To determine the difference over time each of the different trajectories were repeated. For example when the trajectory of the first run reached a radius equal to the original altitude the cycle was repeated. The difference in propellant consumption between the first and second runs shows the value of boosting to a higher altitude. Fuel consumption is dramatically reduced by a large reboost assuming the spacecraft does not have to be at a given altitude at frequent intervals. The small difference between the second run and the check program is harder to explain. The *check* program completes a constant burn at maximum thrust up to an altitude equal to the maximum of the second run, a radius of 1.023 distance units. Mentioned earlier, the second run is not a smooth burn and uses burns during the time normally associated with the decay period. A function that would have contained a single burn to a higher altitude would even result in a lower cost function. For example if the spacecraft would have been boosted an additional 15 km (1.025 distance units) the mass used would have been 0.0336 units with a period of 4261.6 TU. This period approaches the period of the second run with lower propellant consumption. Figure 5-12 shows the comparison with a higher altitude boost. The high boost (radius = 1.025 DU) is clearly the trajectory that would possess the lowest cost function. The question becomes as to why did the optimum code not give an

optimum trajectory. The answer lies in the complexity and size of the vectors involved

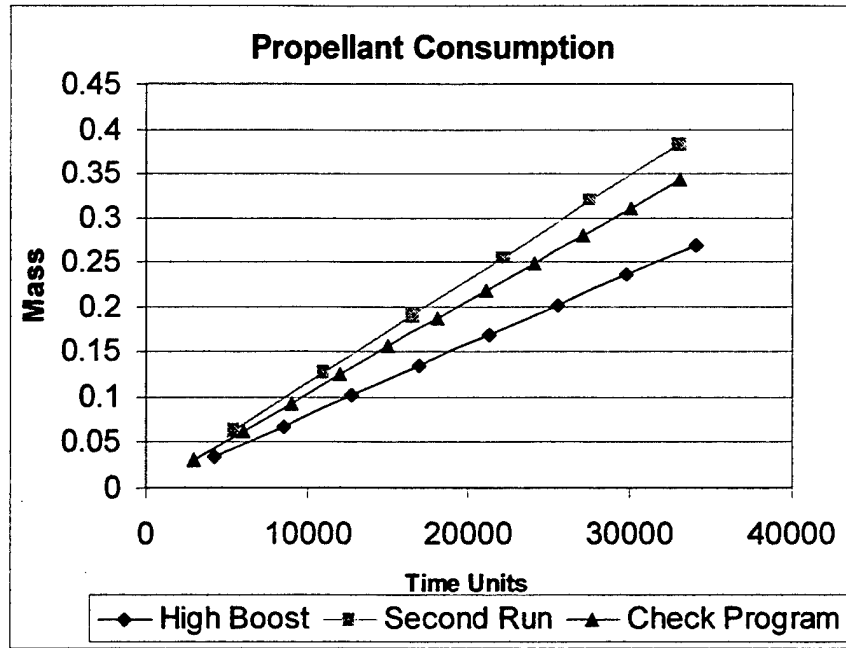


Figure 5-12

with the spectral collocation method, namely, the creation of the state and control matrices and the number of system constraints.

A method to test the optimality of the non-linear code is to run the simulation for a fixed period equal to the period of the free case and compare the results and cost functions. The first run of the free section will be used as the benchmark against which to compare the fixed period simulations. The first run used the full equations of motion and all of the system constraints and gave a very smooth trajectory with a low cost function. The optimal period was 112.6 TU with a cost function of 0.7837. It will soon be obvious that this run is a very unique run in that the combination of guesses for the states, controls, and initial period all allowed the code to iterate to a near optimal solution. In experimentation with the code these "good" runs were very infrequent with problems ranging from matrix scaling problems to solutions that would seem to settle in a local minimum vice the global minimum. In any type of numerical study, reproducibility of results is a key tenet of success. The following runs show the ways the original optimal code was modified in order to reproduce the results of the original run.

The first step in this process is to fix the period at the optimal value obtained from the first run. The codes *orbopt* and *orbcrit* were modified and renamed *tfopt* and *tfcrit*, included in the appendix. The main difference in the programs was the removal of  $\tau$  from the “aop” matrix.

```

aop(7*n+1)=150;    from orbopt.m
tau = aop(7*n+1);

becomes

tau = 112.6

```

Figure 5-13

The change to the *orbcrit* program is similar. Figure 5-14 shows the four states as a result from the period fixed case. The plot of the radius in the top left corner of Figure 5-

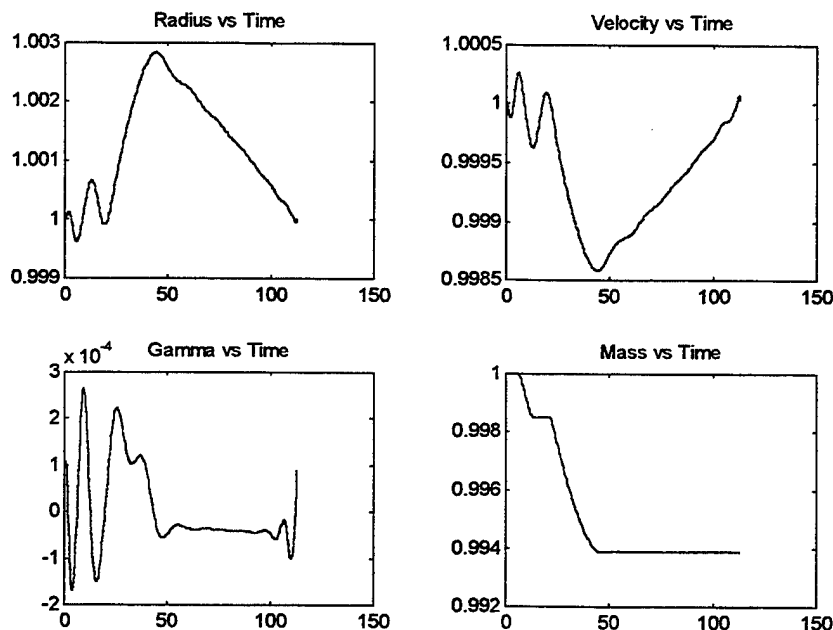
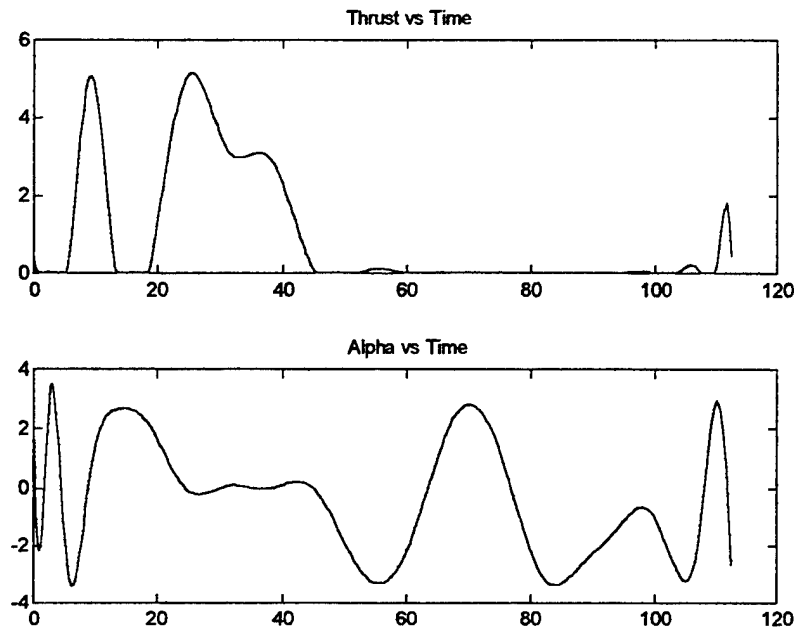


Figure 5-14

14 is very important. Initially the code allows the radius to fall below the initial altitude of 300 km. The general rule of thumb established in previous sections is that fuel consumption will decrease with high altitudes. Conversely, one would expect that lower altitudes require more propellant given the same period. This is exactly the case for this

run. The cost function for the fixed period case is 0.8559 or 9.2% worse than the original run. Figure 5-15 shows the controls for this trajectory.



*Figure 5-15*

The main difference between the control plots in the two runs is the position of the main thruster activity. In the original run the majority of the thrusting action occurred within the first 20 time units. Figure 5-15 shows that the main reboost thrusting is not begun until approximately 20 TU. The initial thrust spike at time 10 is the reboost maneuver that takes the spacecraft that has descended below the original altitude back to its initial height. The final burn is approximately equal to the original run and is required to match the initial and final conditions for the periodic states. The controls for the alpha are fairly analogous to the original run's thrust angles. When thrusters are firing the thruster angles are in the neighborhood of zero, meaning that thrust is usually directed towards the velocity vector.

The results from the fixed period run were disappointing in terms of result reproducibility. The next step was to reduce the number of variables in the non-linear code further to reduce the numerical complexity of the system. The next simplification

affected the angle theta within the equations of motion. Recalling equations (20) and (36)-(39):

$$\dot{r} = \bar{v} \cdot \sin(\gamma) \quad (20)$$

$$\dot{\bar{v}} = -\bar{g} \cdot \sin(\gamma) + \frac{\bar{T} \cdot \cos(\alpha) - \bar{D}}{\bar{m}\bar{B}} \quad (36)$$

$$\dot{\gamma} = \left( \frac{\bar{v}^2}{\bar{r}} - \bar{g} \right) \cdot \frac{\cos(\gamma)}{\bar{v}} + \frac{\bar{T} \cdot \sin(\alpha)}{\bar{m} \cdot \bar{v} \cdot \bar{B}} \quad (37)$$

$$\dot{\bar{m}} = \frac{-\bar{T}}{\bar{v} \cdot \bar{B}} \quad (38)$$

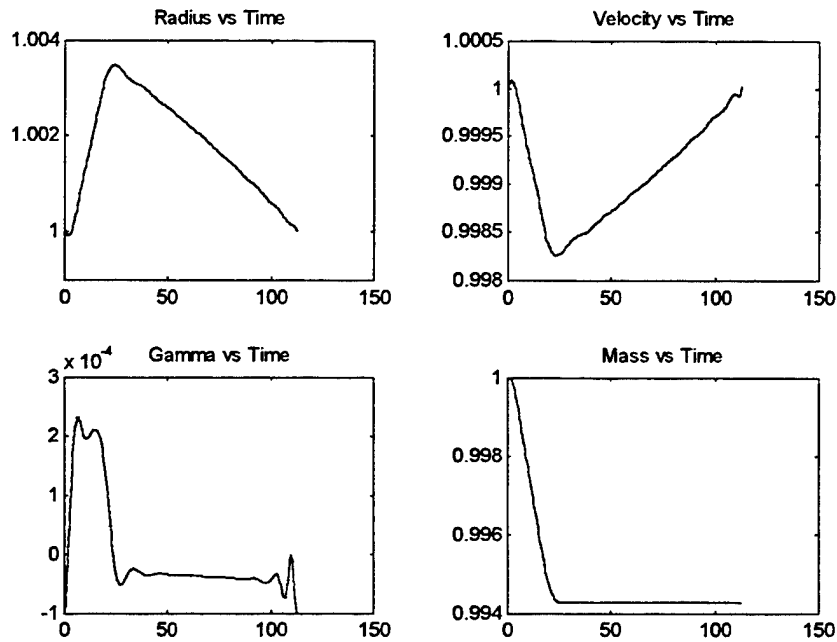
$$\dot{\theta} = \frac{\bar{v}}{\bar{r}} \cdot \cos(\gamma) \quad (39)$$

The equation for  $\dot{\theta}$ , equation (39), determines the differential change of theta over time. This value is not used in any of the other equation of motions and is completely independent of all of the other states. Thus the removal of theta from the system of equations would not effect the other states. The loss of theta from the results does pose some minor problems in analyzing the data but since the values of gamma are very small, on the order of  $10^{-4}$  radians, the angle theta can be approximated by the simulation time. The error in this approximation is significantly less than 1% with an orbit maintenance period of approximately 100. This error only changes the angular position of the spacecraft at any given time and does not effect the magnitude of the orbital radius. The codes *tfopt* and *tfcrit* were modified as shown in figure 5-16 and renamed *notopt* and

<p>Changes to <i>orbopt</i>:  Removed - <code>aop(6*n+1:7*n) = zeros(n,1);</code>  Reduction in the number of constraints –  <code>options(13)=5*n+6 to =4*n+5;</code></p> <p>Changes to <i>orbcrit</i>:  Removed the theta equation of motion –  <code>g(i+4*n) = (2/tau)*sum(Dn(i,:).*</code>  <code>aop(6*n+1:7*n))-(aop(n+i)/aop(i)*</code>  <code>cos(aop(2*n+i)));</code>  Removed constraint - <code>g(5*n+3) = aop(6*n+1);</code></p>
---

Figure 5-16

*notcrit*, both included in the appendix. The effect of these changes reduces the size of the “aop” vector by removing “n” quantities. The removal of constraints also reduces the size of the matrices used by MATLAB. The results reflect the changes with a more optimal run. Figure 5-17 shows the states of the no theta run.



*Figure 5-17*

Figure 5-17 shows plots that are very similar to the original run. The radius is only significantly different in the first few time units. Here the radius is allowed to follow and descend slightly below the original altitude for a very short period. The plot for gamma is consistent with earlier results with a slight increase in the angle of the velocity vector to the orbit tangential during the thruster firing. Figure 5-18 shows the controls for this run.



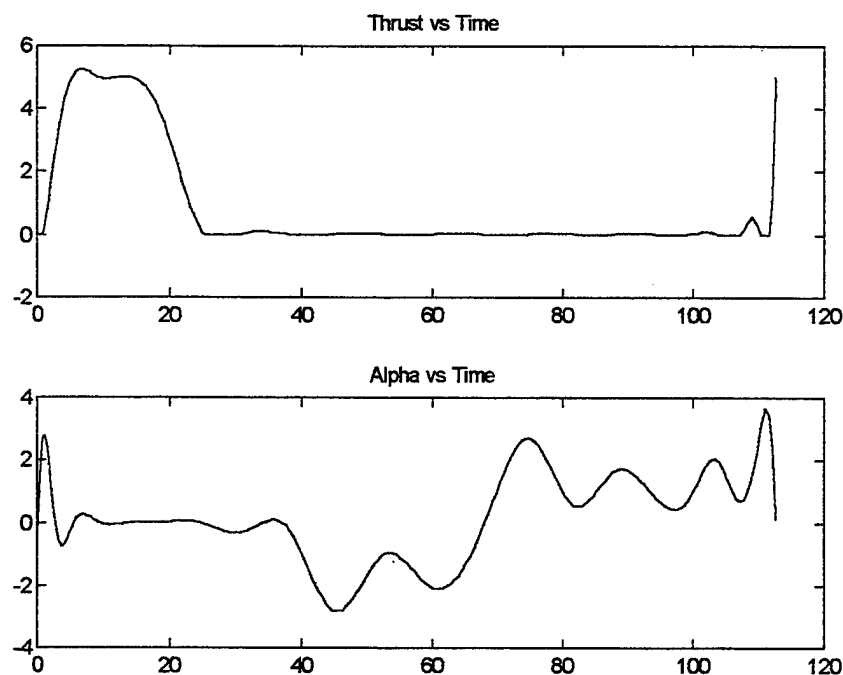


Figure 5-18

Excepting the delay with which the thrust starts its initial burn, the no theta control profiles are nearly identical to the original run. The initial thrust occurs with two distinct maximums and then approaches zero for the duration of the decay period. The final thrust is the characteristic final burn to match initial conditions. The angle alpha is nearly zero for the duration of the thruster burns and wanders freely between positive and negative pi when the thruster is off. The cost function for the no theta run is 0.7945, which is only a 1.4% reduction in optimality from the original run.

The final simplification of the computer code involves the angle alpha. Each of the above cases shows that alpha hovers around zero whenever a thruster is fired. The logical procedure would then be to assume that the alpha is zero always. Physically this would mean that the thruster is constrained to fire in the direction of the velocity vector only. This assumption is commonly used in real world orbit reboost planning and execution. The removal of alpha allows the equations of motion to be dependent upon only one control. Every time alpha appeared in the equations of motion it was included within a trigonometric expression. Since alpha plays a bigger role while the thrusters are firing and the values were around zero, the trigonometric values of these regions were nearly one or zero depending on the trigonometric function. The programs *notopt* and *notcrit* were

exchanged for *noaopt* and *noacrit*, which both included substituting in all instances of alpha with the value zero. In all of the problems alpha was represented by the portion of the “aop” defined as “aop(5\*n+i)”. Figure 5-19 shows the two lines of code in *notcrit*, which contain the alpha term.

```

Ast = (aop(4*n+i)*cos(aop(5*n+i))-D(i))/(aop(3*n+i)*B);
Ant = aop(4*n+i)*sin(aop(5*n+i))/(aop(3*n+i)*B);

The expressions are replaced by

Ast = (aop(4*n+i)*1)-D(i))/(aop(3*n+i)*B);
Ant = 0;

```

Figure 5-19

The size of the “aop” vector is also reduced by “n” terms in the *noaopt* file. The first noticeable difference between the no alpha run and the original run is the length of required computing time. The trimmed code and reduced matrices complete the optimization task in less than one half hour while the original code required over 2 hours to finish. The results are shown below. Figure 5-20 shows the states of the system.

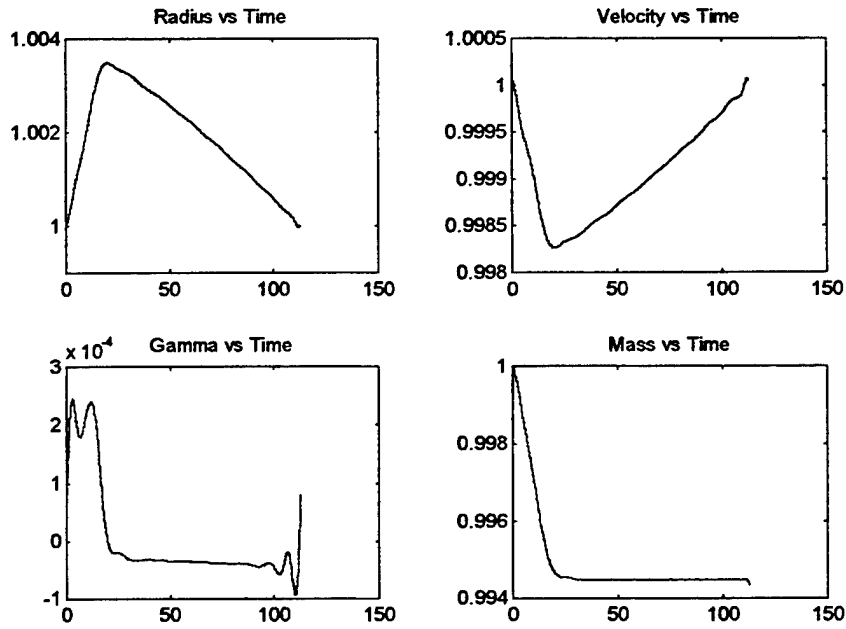
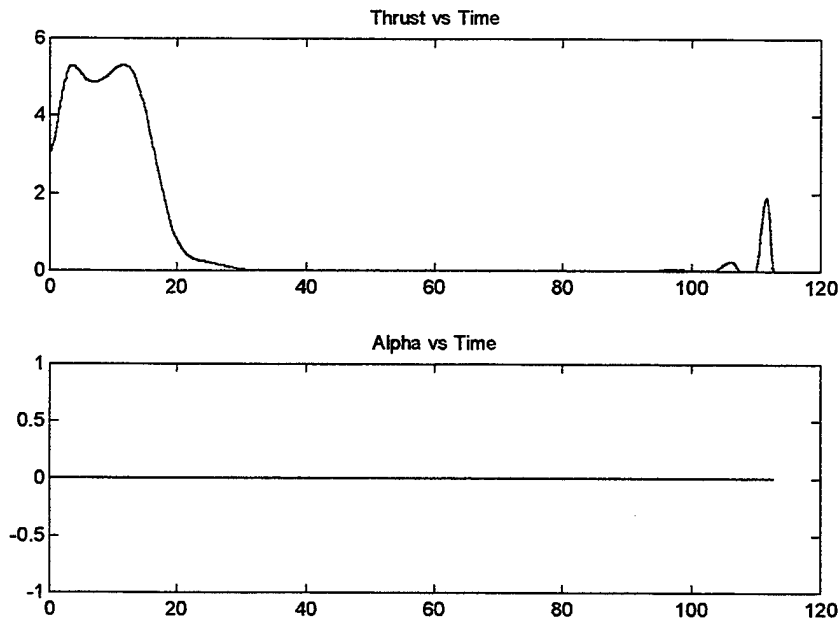


Figure 5-20

These graphs closely resemble the plots earlier shown for the original run. The cost functions are nearly identical with the no alpha case 0.03% more efficient than the original problem. The controls are shown in Figure 5-21.



*Figure 5-21*

While the plot of alpha should be obvious, the plot of the thrust versus the time is almost identical to the thrust plot in Figure 5-4.

The series of modifications to the program produced a code that minimized the effects to the states yet modeled the system with a high degree of accuracy. Figure 5-22 shows a summary of the cost functions for each of the different modification steps.

Method	Cost Function
Full Equations of Motion	0.7837
Fixed Period	0.8559
No Theta	0.7945
No Alpha and No Theta	0.7834

*Figure 5-22*

The initial and final orbits of the optimal code are nearly circular with very small values for the flight path angle. The size of  $\gamma(0)$  and  $\gamma(\tau)$  are equal and are on the order of  $10^{-6}$  radians. The values of the initial and final velocity also differ slightly from the reference value of one, usually in the sixth decimal place. The significance of these differences lies in the fact that the optimal initial conditions require the optimal initial orbit to be slightly elliptical. The no alpha program allows the optimization code have a high degree of reproducibility with fewer errors or warnings given by MATLAB. Another note concerning the no alpha code is warranted. The plots displaying the controls of the program do show a curve for alpha, which is always zero for the no alpha code. The control history of the no alpha optimization code is used in the same first order propagator that the full optimization code uses. Displaying the control history of alpha reminds the viewer that the no alpha optimization code was used. The code also allows the tolerances for the constraints and states to be very small which in turn helps prevent the code from settling in a local minimum without a full series of iterations.

#### **D. BAND FIXED SIMULATIONS**

The next type of trajectory to examine is the case where the spacecraft is bounded by an upper altitude limit. Following the rule of thumb previously established, a spacecraft subjected to this one constraint would follow a trajectory that would maintain the orbital altitude at the upper limit. This would be a thrust-drag cancellation profile and labeled earlier as the high FKT profile. Any flight below the upper limit would require more propellant. While appearing trivial the impact of the preceding statement is significant in that a satellite that possesses a maximum limit can not orbit the Earth with a trajectory that uses less propellant than a high FKT. To illustrate these statements the code was modified to make the initial altitude the upper limit as well. This modification forces the code to optimize be either following a trajectory that goes below the initial altitude or follow an FKT. Figure 5-23 shows the states for the results from the FKT case.

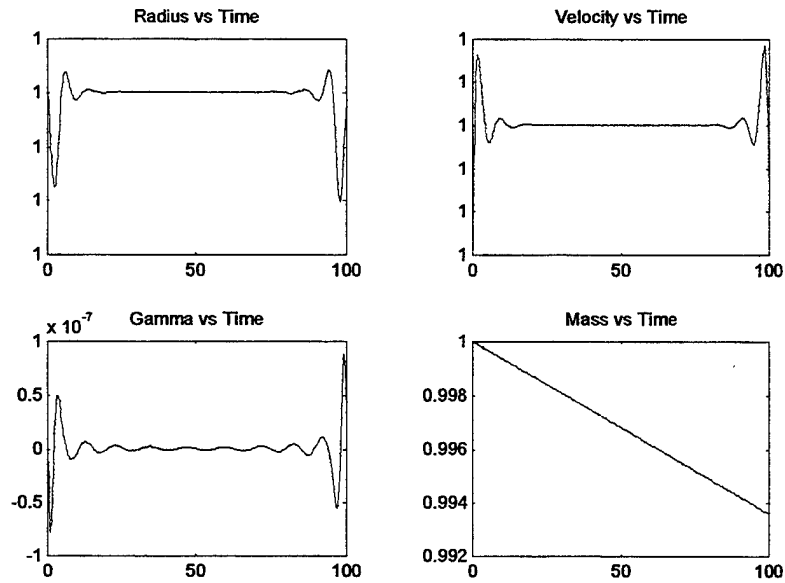


Figure 5-23

While there may appear from the plot alone, that there are large oscillations in the radius during the beginning and end of the period, in actuality the peak of these oscillations has a magnitude of 1.0000001 or  $10^{-7}$  deviation from the state. The  $10^{-7}$  is the tolerance the optimal code uses in verifying the validity of its states. Thus neglecting these small numerical oscillations the radius follows an FKT profile. Figure 5-24 shows the controls for the FKT case.

Once again neglecting the numerical inaccuracies at the very ends of the simulation the

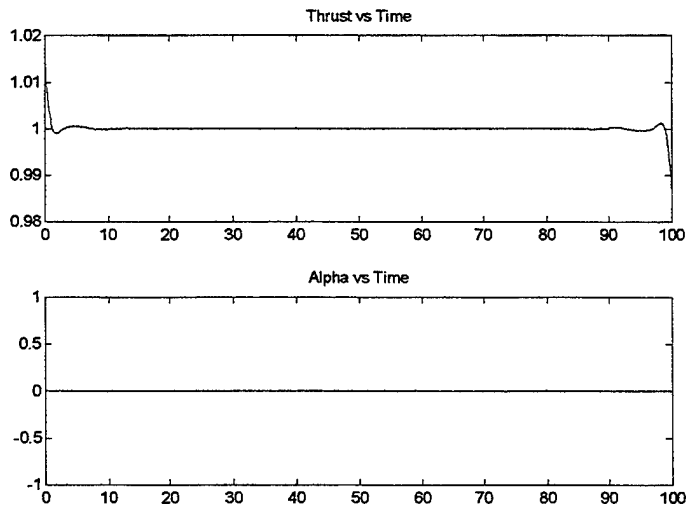


Figure 5-24

value of the thrust is a constant one. Since thrust is defined in terms of drag, a thrust value of one means that thrust equals drag. The satellite flies an FKT profile.

However very few satellites today have the capability to follow a thrust-drag cancellation profile. Continuous or long duration burn thrusters are only beginning to be seen with the advent of electric propulsion systems. So most mission planners use the orbital band for planning the orbit maintenance schedule for a vehicle. To look at this type of mission the program first fixes the initial altitude and since the radius is periodic the final radius is also defined. The next parameter to examine is the period  $\tau$ . If  $\tau$  were free, the satellite would boost from its lower altitude to its upper limit and then begin a high FKT for an infinitely long time. Since this defeats the purpose of the mission planner a  $\tau$  needs to be chosen. The optimal code is given a fixed  $\tau$  as in the earlier examples and fixes the initial altitude at an initial height. The only additional parameter is a value for the width of the orbital band, labeled "band" in the program *noaopt*. Figure 5-25 shows the

$$g(i+6*n+5) = aop(i)-(1+band);$$

Figure 5-25

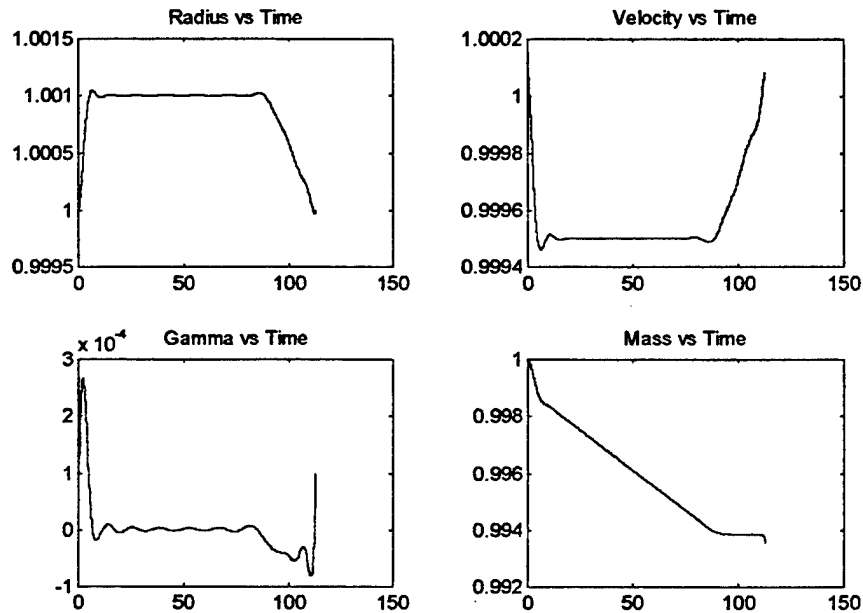
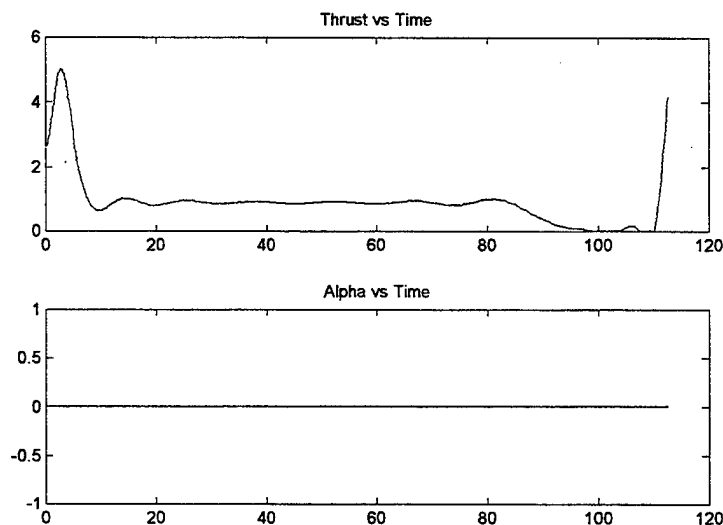


Figure 5-26

additional inequality constraint added to the *noacrit* program to bound the upper altitude.

The first run was with the same period as before, 112.6 time units. The band was entered in as 0.001 DU or about 6.7 km. The initial altitude remained at 300 km and the generic spacecraft was the vehicle modeled. The no alpha code was used due to its numerical reliability and shorter computational time. The states for the first run are shown in Figure 5-26.

The graph for the radius in the upper left shows the spacecraft follow a direct path to the high altitude limit, then begin an FKT trajectory, followed by decay to its original altitude. This combination of regimes can also be seen in the mass plot. Initially the rate of propellant use is large during the initial burn. The slope of the mass curve changes when then spacecraft reaches its upper altitude limit to a lower rate, consistent with a thrust-drag cancellation burn. The next change in the mass' slope occurs when the slope becomes zero, during the decay phase when propellant is not used. The final change in the graph's slope is due to the final burn required to match the initial conditions. Figure 5-27 shows the controls for this case.



*Figure 5-27*

The thrust begins with a large initial burn with a gradual taper to a thrust approximately with the value of one. The thrust ends with the characteristic final burn.

Numerous runs were made to examine to verify that the type of profile seen in the above example was consistent with varying orbital bandwidths and orbit maintenance periods. One should note that the constraint placed on the maximum radius was not

always used. If the band was sufficiently high or the period fairly short the satellite never made it to the point where it began an FKT profile. Instead the plots were similar to the plots shown in the fixed tau cases when the radius was free and unbounded. Whether or not a spacecraft's radius was constrained depended on the period of the case. If the boost time and decay time filled the entire period without boosting the satellite to the constraint height an unbounded trajectory was the result. For completeness another example is shown. In this case the band is 0.005 distance units, just over 33 km. The period is 400 time units. Figure 5-28 shows the states of this run.

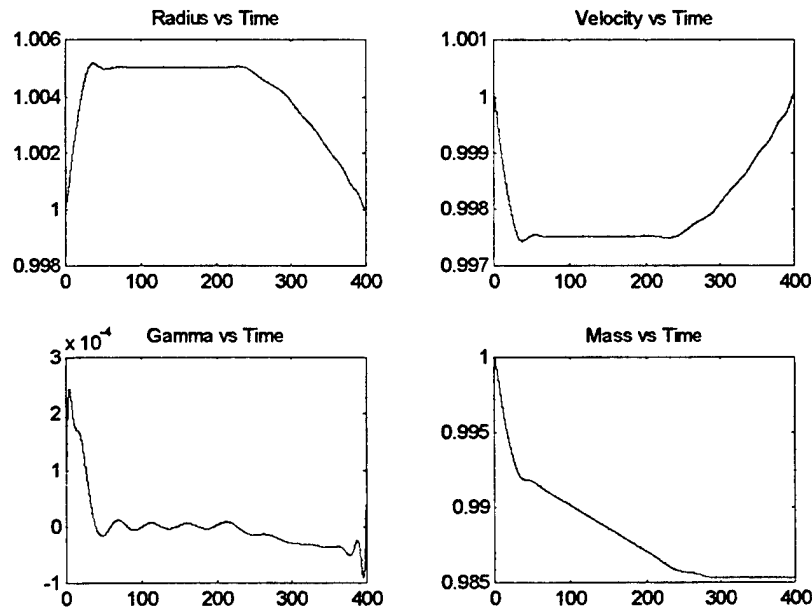
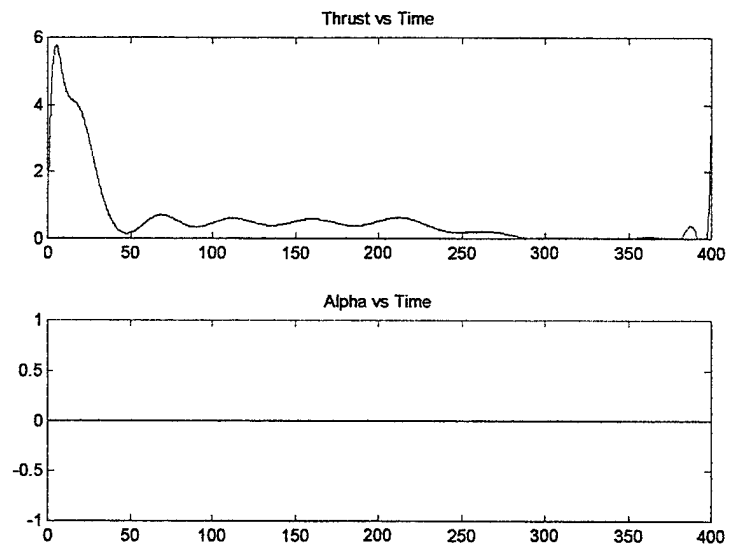


Figure 5-28

Figure 5-29 shows the controls of this example. Both figures resemble the state and control figures from the previous example. It is a reasonable conclusion that this is the shape of the optimal trajectory given the noted system constraints.





*Figure 5-29*



## VI. ANALYSIS OF OPTIMALITY

### A. OPTIMALITY TEST CODES

Previous sections have examined the optimal periodic problem, the numerical method and the implementation of the non-linear problem. This chapter is devoted to using the developed tools and examines their performance for varying different parameters. A measure of the performance is to analyze the optimality of the different solutions by comparing them to conventional maneuvers. A series of programs use the *orbprop* propagator to develop trajectories that simulate maneuvers that are compared with the results of the optimal code. The first of these is called *hoh* and models the Hohmann transfer. This program is given an orbital bandwidth. It then calculates the change in velocity required to travel from a circular orbit at the lower altitude to another circular orbit at the higher altitude. The Hohmann maneuver uses impulsive burns, meaning that the burns are instantaneous. To model this a velocity change is added directly to the vehicle when it reaches the bottom of the orbital band. At the same time the velocity is added, the mass used in the impulsive burn is subtracted from the vehicle weight. Figure 6-1 shows the lines of code which control the additions of velocity and subtractions of mass.

```
First Impulsive Burn
vfa = (1/xx(k,1))^0.5;           %Lower Circular Velocity
vtxa = (2/xx(k,1)-1/atx)^0.5;   %Transfer Velocity at A
dv = abs(vtxa-vfa); vn = xx(k,2)+dv;
mf = xx(k,4)*exp(-dv/ve);      %New Mass After Burn

Second Impulsive Burn
vtxb = (((2/(r+band))-(1/atx)))^0.5; %Transfer Velocity at B
vfb = (1/(r+band))^0.5;         %Higher Circular Velocity
dvb = abs(vfb-vtxb);
mf = xp(z,4)*exp(-dvb/ve);     %Final Mass after Transfer
vn = dvb + xp(z,2);            %Velocity addition
```

Figure 6-1

The Hohmann program propagates the orbit over the given period. When the radius falls below the lower altitude limit, the Hohmann routine transfers the vehicle to the upper altitude. The mass and time of the Hohmann burns are recorded and then written to a data file for further use.

The programs *bchk* and *check* are very similar and simulate a continuous burn type trajectory. The program *check* is given a maximum value of thrust. At the start of the simulation the vehicle is placed into a maximum tangential burn which continues until the vehicle breaks the top of the altitude band. The five physical states are then interpolated to give a specific time the spacecraft crossed the altitude upper limit. The next part of the program calculates the decay time from the top of the band to the lower limit. The times are added together to form a final period. The values for the mass and times for the one tangential burn method are recorded for latter comparison. The program *bchk* operates in much the same manner with a minor exception. Here the program is given the entire period of interest beforehand. For example to compare the *bchk* trajectory versus the optimal code the same optimal time period  $\tau$  would be used. The program calculates the thrusting boost time and the decay time. The burn time and decay time are added and

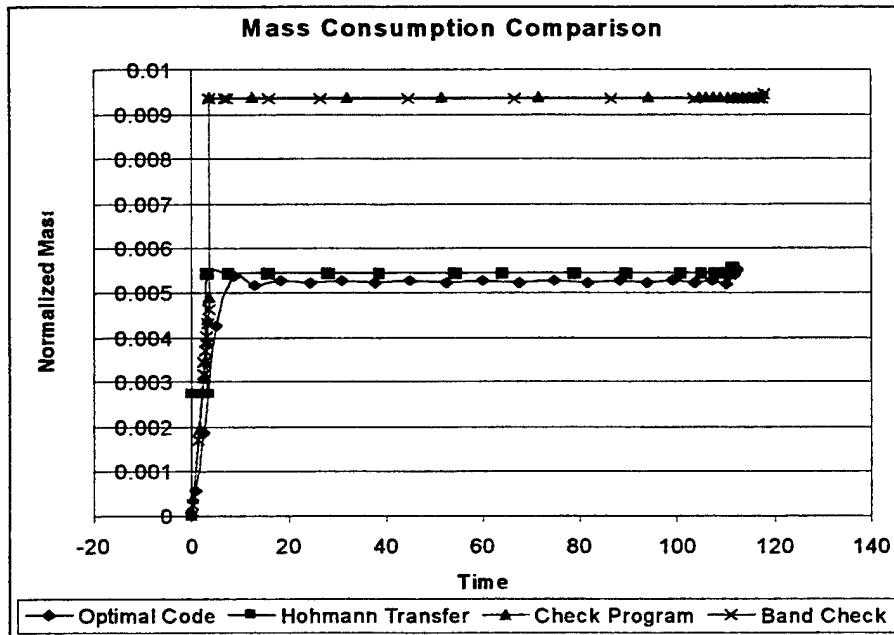


Figure 6-2

subtracted from the given period. The time remaining is filled with a high FKT at the

upper altitude limit. The trajectory resembles the band-limited trajectories included in the last chapter. Here also the values of the masses and times are recorded for comparison.

Figure 6-2 shows the differences in the mass plots of the four different trajectory programs. The two check programs require considerably more fuel than the Hohmann and Optimal trajectories. The codes are meant to mimic a real world continuous thrust to the upper altitude limit. The trajectories are very costly in terms of propellant due to the fact of the burn required to circularize the orbit to the upper altitude limit. There are two reasons that this needs to be accomplished. The first is that the upper circular orbit is the same final orbit after the Hohmann transfer. The second reason is that the angle  $\gamma$  is a positive number, meaning that the velocity vector is above the local horizontal. After the thruster is turned off the spacecraft would travel beyond the upper altitude constraint in an elliptical orbit. Figure 6-3 is a diagram showing the necessary change of velocity required to circularize the final orbit at the upper altitude limit.

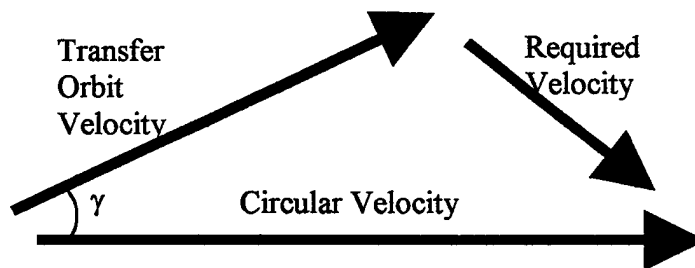


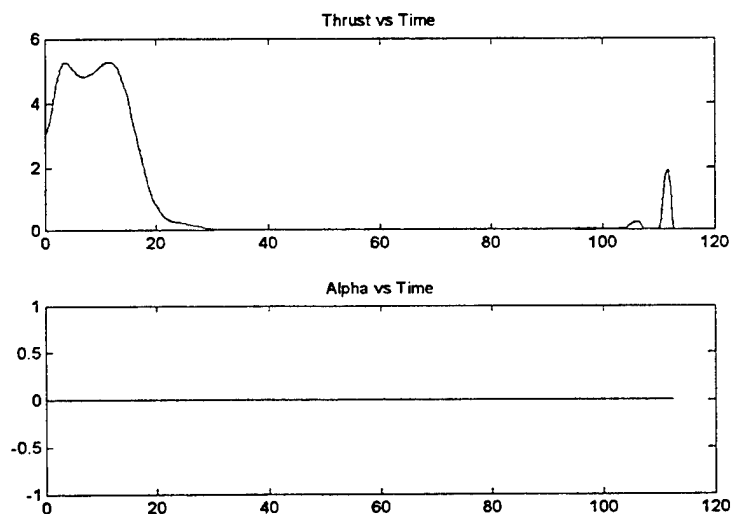
Figure 6-3

For low values of thrust, where  $T = 1$  to 5 normalized thrust units, the required  $\Delta v$  is fairly small. When thrust increases,  $\gamma$  increases lengthening the required velocity change vector. Thruster sizes that approximate instantaneous burns require a large second burn to circularize the final orbit. The thruster size used in figure 6-2 for the *check* and *bchk* programs was set at 20 normalized thrust units. All four programs will be examined later to measure each trajectory's optimality.

## B. VARIATION OF PARAMETERS

The true test for the code is its ability to vary the initial guesses, control boundaries, numerical method parameters and deliver reliable and accurate results. This is especially

crucial for optimal control problems that are very sensitive to initial values and effectiveness of the controls. In the previous chapters different values and equations were tested to achieve good results. Throughout these examples, thrust has been constrained to never go above a certain value. In this case thrust has been constrained with a value of 5 normalized units. Physically the value of thrust changes with the location and physical attributes of the spacecraft. For example for the generic spacecraft at 300 km altitude, one normalized unit of thrust equals 0.7 N while the same normalized unit for the ISS at the same altitude is 2.9N. The engine planned for the ISS is 110N or just over 37



*Figure 5-19*

normalized units at the reference altitude. Therefore it is beneficial to study the effects of increased thrust constraints and eventually the unconstrained thrust case.

Recalling Figure 5-19, the controls of the generic satellite given a fixed tau of 112.6 TU, the plot for the thrust shows the smooth continuous thrust history with the initial thrust defined by two characteristic peaks with a maximums around 5.0. Note that the reason the graphs appear to slightly break the upper thrust constraint are due to the fact that the plots represent an interpolation of the values taken from the values of the controls determined by the optimization code at the LGL points. In attempting to smooth the curve the interpolation routine will go above the stated constraints. However the shape of the curve is still a good approximation of the behavior of the system. With this thrust constraint the cost function for the given period was 0.7834. The propellant consumed

during the simulation was approximately 0.0057 normalized units or in physical terms just under 17 kgs.

Thrust was then increased to a value of 10 thrust units. This must be done in two places within the computer code. First in the *noaopt* code the initial thrust vector was set to 10 units. In the *noacrit* code the inequality constraint containing “*aop(4\*n+i)*” was given a maximum limit of 10. Figure 6-4 shows the control history of the new simulation. This figure shows another smooth function for the thrust but with a single peaked initial

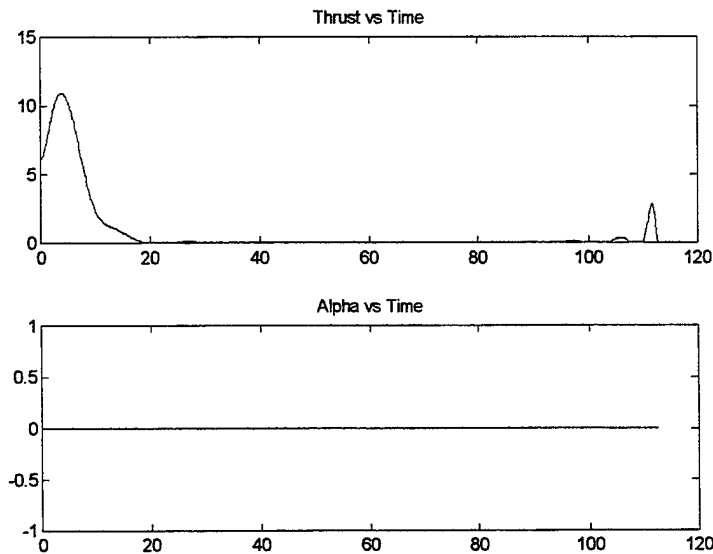
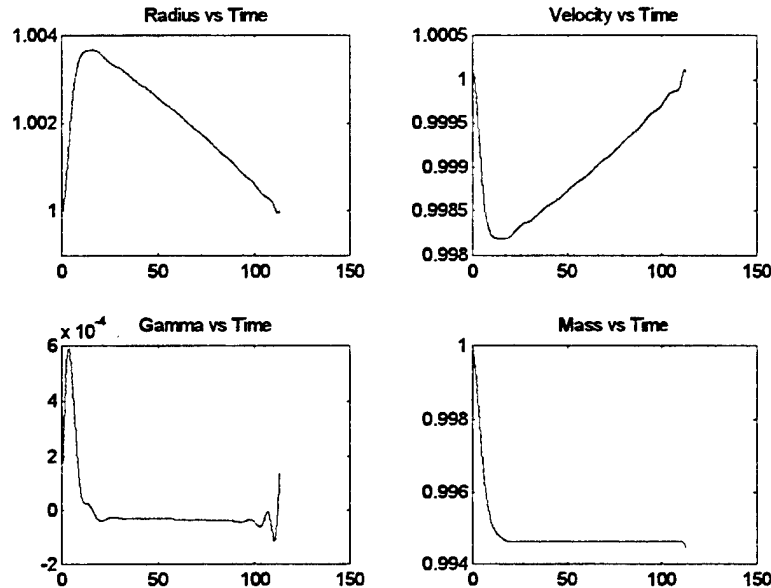


Figure 6-4

burn. The final burns at the end of the simulation are about the same for both cases and do not reach the maximum thrust value. The cost function for this case is 0.7688, a 2% reduction in fuel consumption. In terms of physical mass of fuel saved by having a higher limit of thrust, it represents 0.3 kg of savings. This may not seem like a substantial amount but recall that the period for this case is slightly longer than a single day. Increasing the motor size from 3.5 N to 7 N would save the spacecraft approximately 100 kg of propellant needed for orbit maintenance over the course of a year. The reason for the fuel savings is that the larger motor is able to boost the satellite to a slightly higher orbit with the period constraint. The 5 thrust unit case boosted the satellite to a normalized altitude of 1.0034 DU while the 10 thrust unit case boosted it to 1.0036 DU. This change in altitude difference was enough to reduce the cost function ever so slightly.

The shorter burn time to reach the higher altitude allows longer decay times. The basic shape of the trajectory is very similar to the earlier case. Figure 6-5 shows the states for the 10 thrust unit example.



*Figure 6-5*

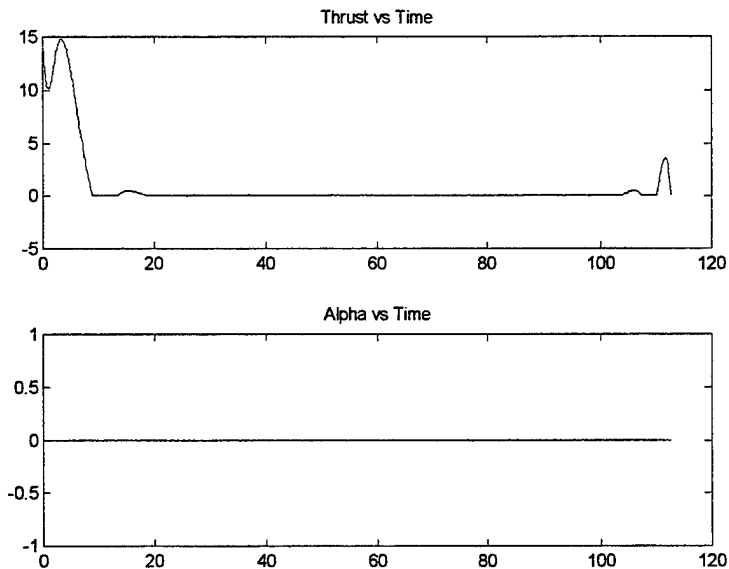
The states have the same shape as those of the lower thrust case, so the addition of thrust does not alter the trajectories significantly.

Since the thrust history was constrained at the maximum thrust limit during the last simulation, another run is necessary with higher thrust levels. The next simulation used a maximum available thrust of 25 thrust units (14 N). The ultimate goal of raising the thrust constraint is to find a case where the maximum thrust does not reach the constraint; in this sense, the thrust would be unbounded. An easy option would be to set the thrust limit at a very high number in the hundreds or thousand. Unfortunately the code sees the large range of available thrusts as an added complexity for the system, leading to non-smooth and non-optimal solutions. Thus an incremental approach to raising the altitude limit is desired. The controls for this run are included in figure 6-6.

The controls from the T=10 and T=25 examples are very similar. The most significant point of figure 6-6 is that maximum thrust is 14.3 thrust units. This is the first case where the thrust did not reach the maximum constraint with an optimal trajectory. The cost function of this run is 0.7620. This cost function is a 3% reduction over the original thrust



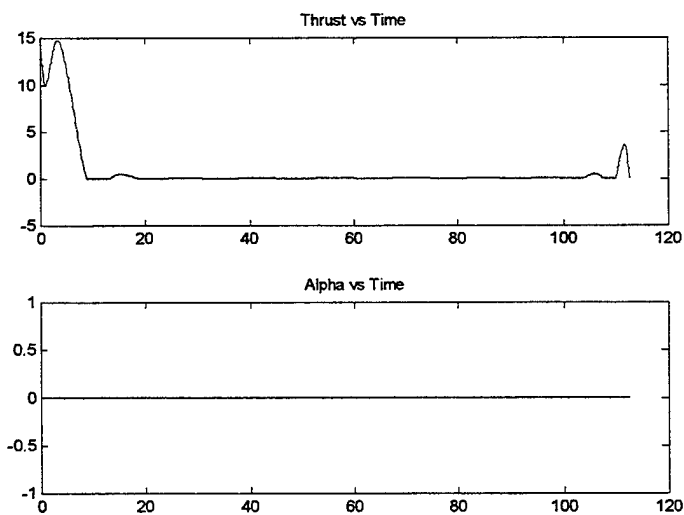
constrained case. To confirm that this solution is not a numerical accident another case with a different thrust constraint ( $T=20$ ) was run. It is important to determine that the optimal thrust profile is not a function of the location of the thrust constraint or initial



*Figure 6-6*

guesses for the thrust history. The cost function of this second run was also 0.7620. All of the states were the same with minor fluctuations in the thrust history in the 5<sup>th</sup> decimal place.

Figure 6-7 shows the controls for the  $T=20$  run.



*Figure 6-7*

The thrust functions are identical between the two runs. The states are included in Figure 6-8.

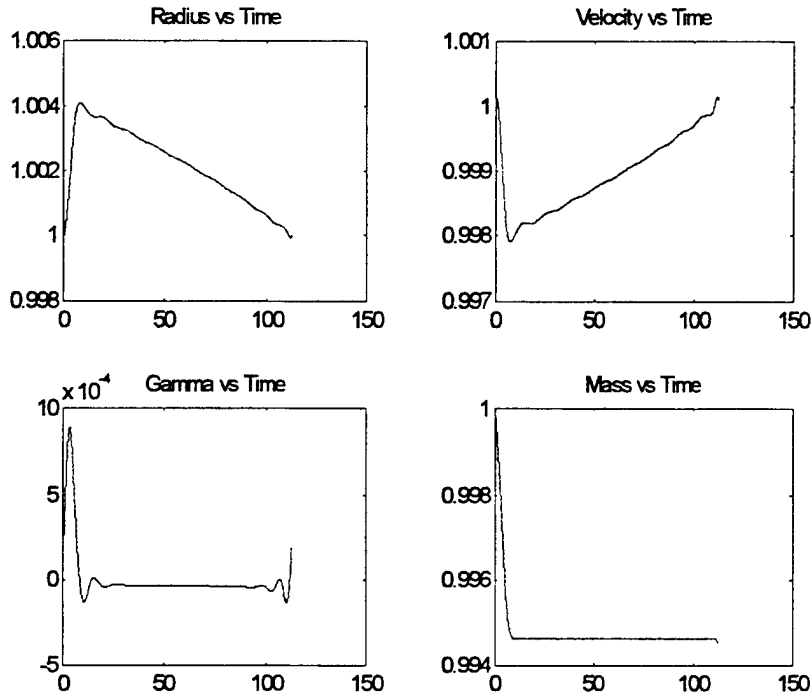


Figure 6-8

The states shown in Figure 6-8 represent the optimal solution given the generic spacecraft, starting at the reference altitude and given a certain period. This procedure can be repeated for any variations of the given information. However, the states shown in Figure 6-8 represent the optimal paths for this unique situation. To develop other situation only requires changing the conditions under which *noaopt* and its associated codes run. For example a different set of curves can be determined if the satellite was to have a two and a half day orbit maintenance period.

The next step is to check the results of the optimal code against the Hohmann trajectories given the same system constraints. One of the tenets from the beginning of the study was to identify a trajectory that was as good or slightly better than an ideal Hohmann trajectory. Figure 6-9 shows the mass consumption of various profiles including the optimal code run with  $T_{\max} = 20$ , the Hohmann transfer and several FKTs.

The FKT paths represent, from the top to bottom in Figure 6-9, the low-band, mid-band,

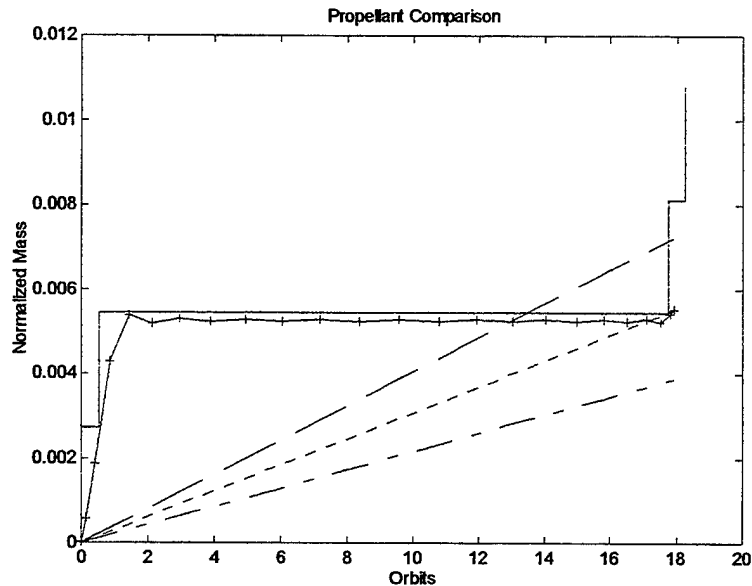


Figure 6-9

and high-band trajectories. As noted earlier the high-band FKT will always be the trajectory with the lowest fuel consumption but the figure assumes that the satellite begins the simulation at the top of the band, neglecting the boost maneuver needed to reach the top of the band. The straight stair-step like solid line represents the mass consumption of the Hohmann transfer. The line with the plus signs represents the mass consumption of the optimal trajectory. Notice that optimal code shows lower propellant use. In order to highlight these differences the graphing periods are increased to show savings over longer periods of time. Figure 6-10 shows the extended mass consumption comparison for the constrained thrust example.

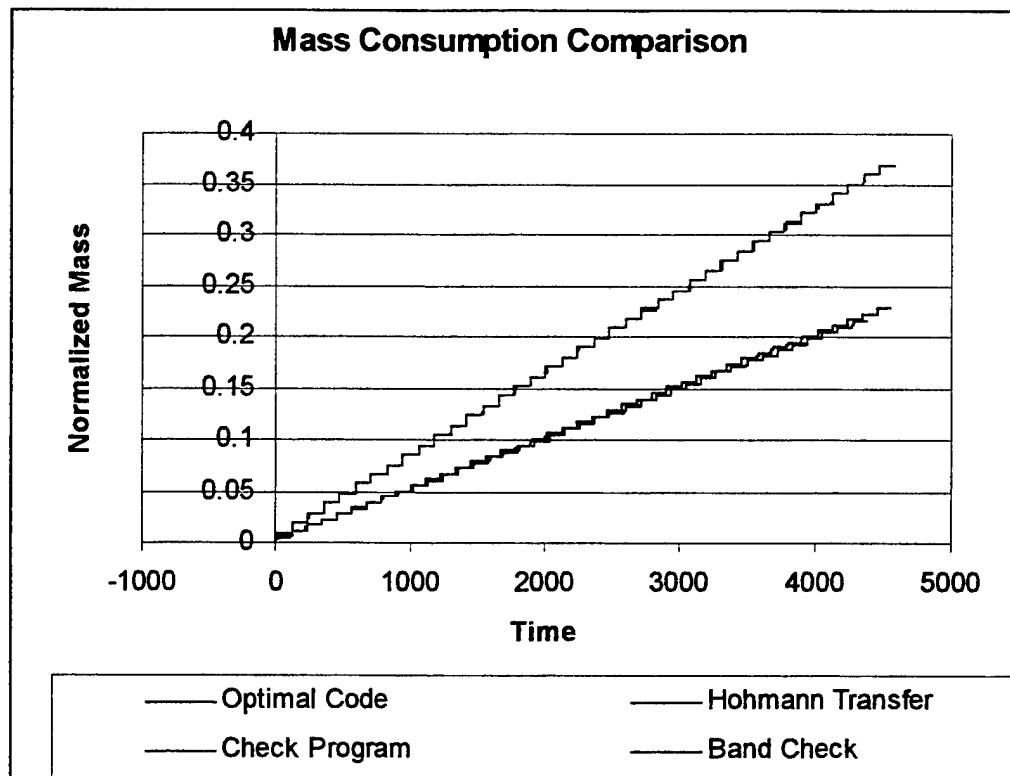


Figure 6-10

The uppermost line shows the propellant consumption for the two check programs. In this instance both programs show the same trajectory. This is because the time to decay from the top of the band is greater than the given orbit maintenance period. When the decay time is subtracted from the given period the result is a negative number. The *bchk* code considers this value zero; in other words the time that the satellite follows a high-band FKT is zero. The next two lines are very close together but a difference is discernable. The higher of the two lower lines is the propellant use of the Hohmann trajectory. The optimal trajectory has the lowest rate of fuel use of the four trajectories. In order to highlight the differences between the optimal and Hohmann trajectories a trendline is extended from each set of points. Figure 6-11 shows the trendlines for a period of approximately 150 days.

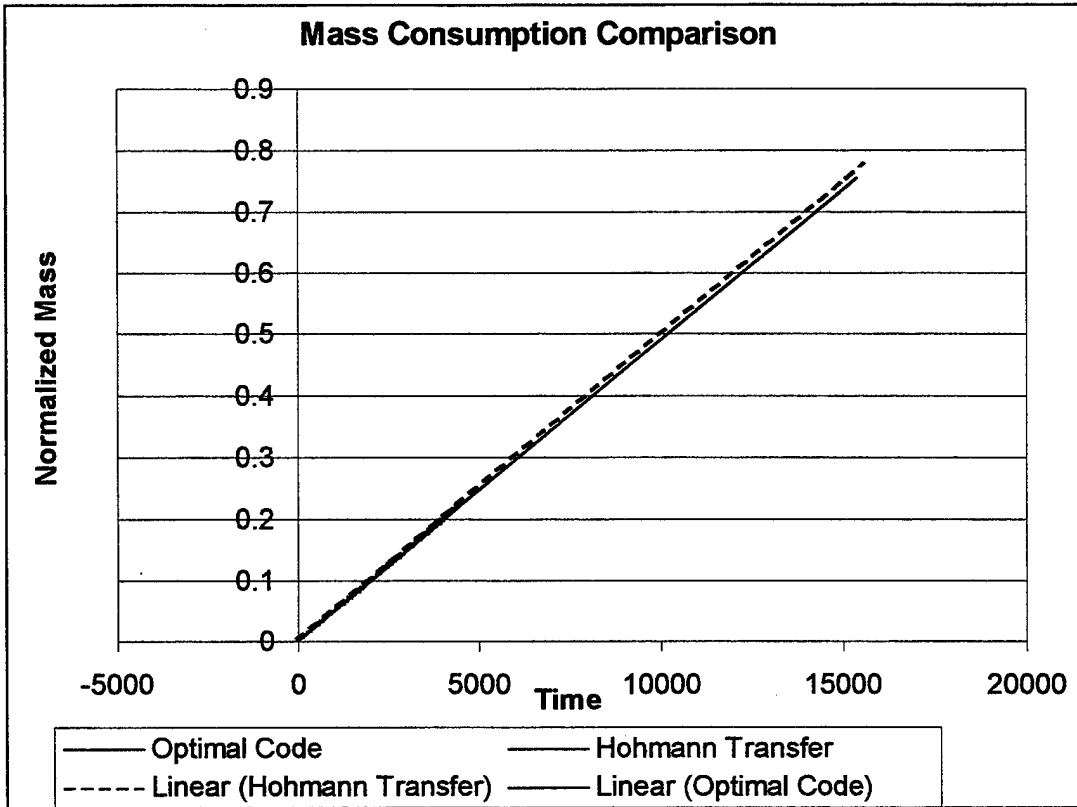


Figure 6-11

This graph shows that one of the goals of this project has been met. There exists a trajectory that is more fuel-efficient than a Hohmann transfer. While the difference between the two is small one must remember that the Hohmann transfer was modeled as an ideal maneuver. The burns are instantaneous with infinite thrust. This assumption neglects losses, mainly drag loss, which the optimal control code considers automatically.

The easiest way to show the effect of drag loss is to start with Newton's equation of motion with the assumption that mass changes are small.

$$F_{net} = ma \tag{76}$$

Assuming constant acceleration equation (76) becomes:

$$\frac{F_{net} \Delta t}{m} = \Delta v \tag{77}$$

$$F_{net} = T - D \tag{78}$$

Thus the required  $\Delta v$  can be related to the total impulse through equation (77). Recalling a form of the rocket equation,

$$m_p = m_o \cdot \left[ 1 - e^{\frac{-\Delta v}{I_{sp} \cdot g_o}} \right] \quad (79)$$

For a real motor:

$$\dot{m} = \frac{F}{g_o I_{sp}} \quad (80)$$

Assuming constant thrust:

$$\dot{m} = \frac{m_p}{\Delta t} \quad (81)$$

Here  $\Delta t$  is the burn time of the thruster. Combining equations (80) and (81) gives an expression for total impulse in terms of propellant mass.

$$F \Delta t = \frac{m_p}{g_o I_{sp}} \quad (82)$$

Assuming that drag works in opposition to thrust, equation (82) can be rewritten.

$$T \Delta t - D \Delta t = \frac{m_p}{g_o I_{sp}} \quad (83)$$

The  $D \Delta t$  term from equation (83) is the portion of the total impulse lost to drag. Since the total impulse must remain the same for a given change in velocity, the drag impulse must be made up by the thrust impulse. This either requires a larger thruster or a longer burn time. For example, assume that a total impulse of 200 units is needed. With a thruster with a normalized thrust of 2, or twice the force of drag, the spacecraft would need a thruster firing of 100 time units to accomplish the maneuver if there were no drag. With drag, the net force is equal to one unit and requires a burn time of two hundred time units. The extra time required for the maneuver determines the amount of propellant required. The addition of drag uses twice as much propellant, or in other words the drag loss of this example is approximately 50%. With larger thrusters the drag loss is decreased. However even for a motor that has a rating of 15 normalized units, the optimal thruster example determined previously, nearly 7% more propellant is required by considering drag. Figure 6-12 modifies the Hohmann propellant consumption of figure 6-11 by adding a 6.67% increase in the amount of fuel required.

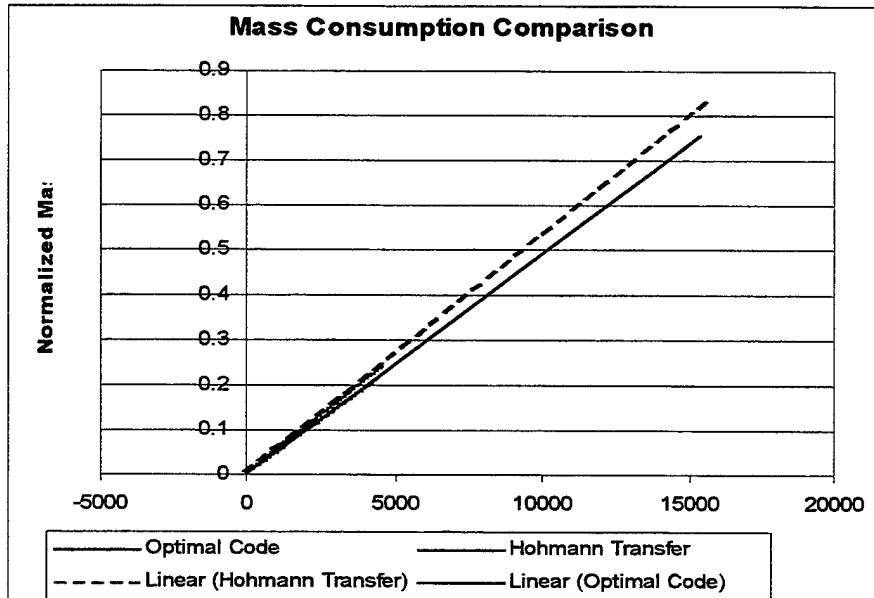


Figure 6-12

Figure 6-13 is included in order to show numerical values for the fuel savings of the optimal code.

	Mass Expended	Percent Savings	Launch Cost Savings
Optimal Code	644.3 kg	-	-
Ideal Hohmann	667.0 kg	3.5%	~\$250,000
Hohmann with Drag Loss	711.5 kg	10.4%	~\$740,000

Figure 6-13

The sampling point for Figure 6-13 was at approximately 45 days of orbit maintenance operations. The thruster size was set at the maximum of the optimal code, 15 normalized thrust units. The 3.5% savings the optimal trajectory has over the impulsive Hohmann by itself is a significant improvement over the life of the satellite. When drag effects are taken into consideration the savings improve dramatically. The general rule remains that the smaller the orbital maintenance thruster the greater the savings of the optimal code

compared to the finite-burn Hohmann transfer. The above example shows that 10.4% savings in propellant can be achieved when the size of the motor equals 15 thrust units (10.5 N). The efficiency of the Hohmann transfer can be improved by increasing the size of the orbit transfer engine, which decreases the burn time. However even at the maximum limit, an infinite thruster, the Hohmann transfer is still 3.5% less efficient.

When monetary terms are used to describe the potential savings the differences become more evident. Looking at Figure 6-13, the savings between the Hohmann and the optimal code is 23.3 kg of fuel. Using an average launch cost to LEO this comes to approximately \$256,000. When drag losses are considered, the difference increases to 67.2 kg, or in terms of dollars \$740,000. The generic satellite used in the example is a high drag vehicle that operates at a fairly low altitude. However, savings occur with other spacecraft as well. For example, the mass/area ratio was changed to model the ISS during a late stage in construction. Figure 6-14 shows the physical characteristics of the ISS example.

ISS Characteristics	
Orbital Altitude	300 km
Mass	408,420 kg
Area	2200 m <sup>2</sup>
Drag Coefficient	2.35

*Figure 6-14*

One rather large difference between the future ISS operations and the optimal code is that the length of time between rendezvous for the ISS is much longer than the period entered into the optimal code. The main reason for this is that long periods create some numerical instability for the optimal code without extensive gradient information. The example though does highlight some of the benefits of an optimal trajectory for shorter time periods versus the mission planned with Hohmann type transfers. All that was changed from the generic satellite to the space station example were the constants for



mass and area, included in the beginning of the *noaopt* program. A maximum thrust of 20 was placed as the initial guess for the code, which for the ISS is approximately 57.7 N. The period of the optimal code was fixed at 112.6, the same period as previous examples.

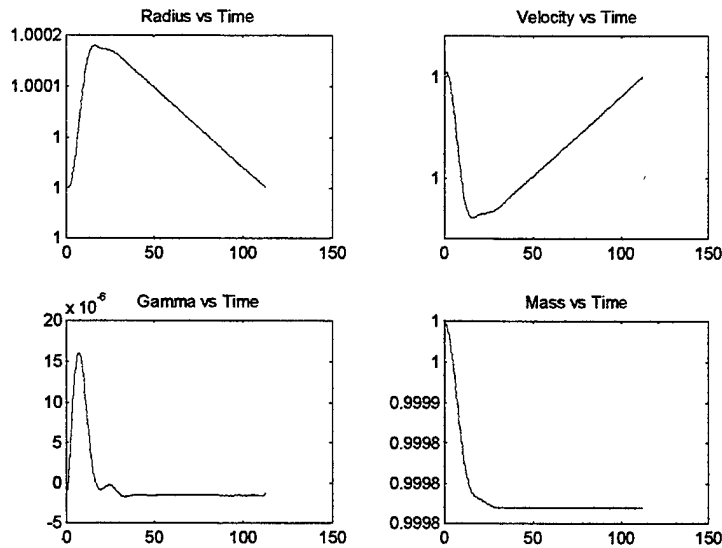


Figure 6-15

Figure 6-15 shows the states for this example.

The plots for the states are nearly identical in shape to the states of the generic satellite. The difference lies in the values of the states. The ISS is much more massive than the generic vehicle and has a normalized ballistic coefficient two order of magnitudes larger.

Figure 6-16 shows the controls for this run.

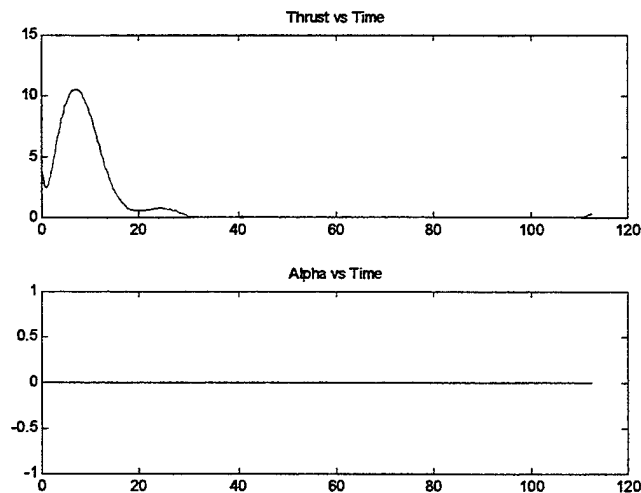


Figure 6-16

Again Figure 6-16 does not possess any large deviations from earlier plots. One interesting point to notice is that the maximum thrust for this case is just over 10 units (or about 30 N). For this case and period the optimal thruster is much smaller than the thruster currently being planned for the ISS, by approximately a factor of four. Figure 6-17 compares the Hohmann transfer fuel consumption versus the optimal trajectory.

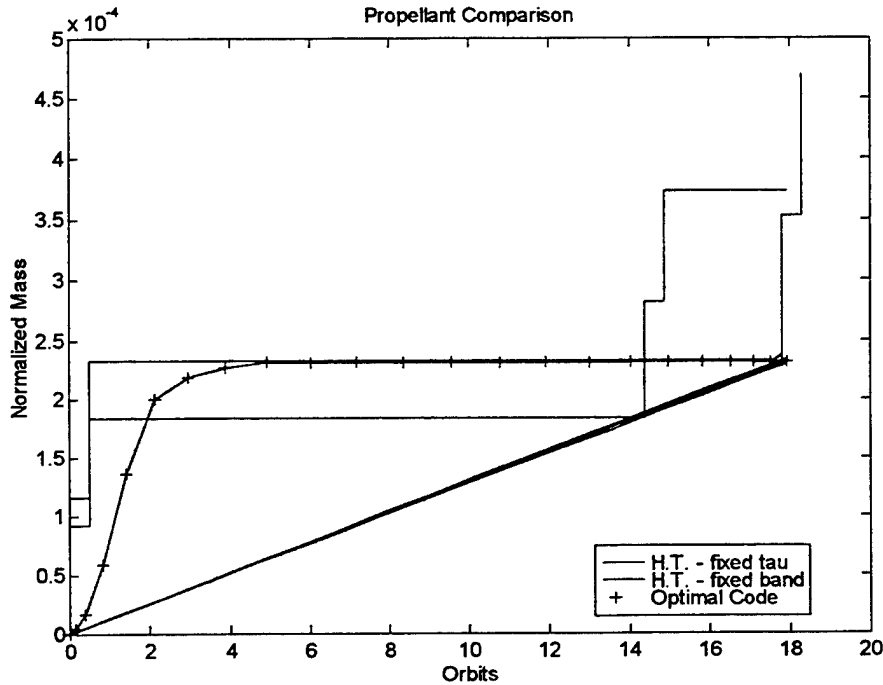


Figure 6-17

The optimal code is the hash-marked line that increases smoothly as the vehicle undergoes its transfer burn and then is constant during the following decay. The figure also shows two different Hohmann transfer plots. The solid stair-step line is the mass consumption of the Hohmann transfer when the transfer uses the same size orbital band as the optimal code. The dashed stair step line is a Hohmann transfer where the period of the transfer is the same as the period of the optimal code. The straight lines are the three different FKT profiles. Since the band in this case is very narrow (about 1.5 km) there is little difference between the high-band to low-band FKT. The fixed tau Hohmann transfer nearly matches the optimal fuel efficiency until the final circularizing burn at the end of the period. At first glance Figure 6-17 suggests that the fixed band Hohmann transfer uses less propellant

than the optimal code. However, the period of the fixed band Hohmann transfer is shorter than the optimal trajectory. In order to examine the difference between the two orbit transfers the values of each are linearly extrapolated over a period of 150 days. This highlights long-term divergence between the two methods.

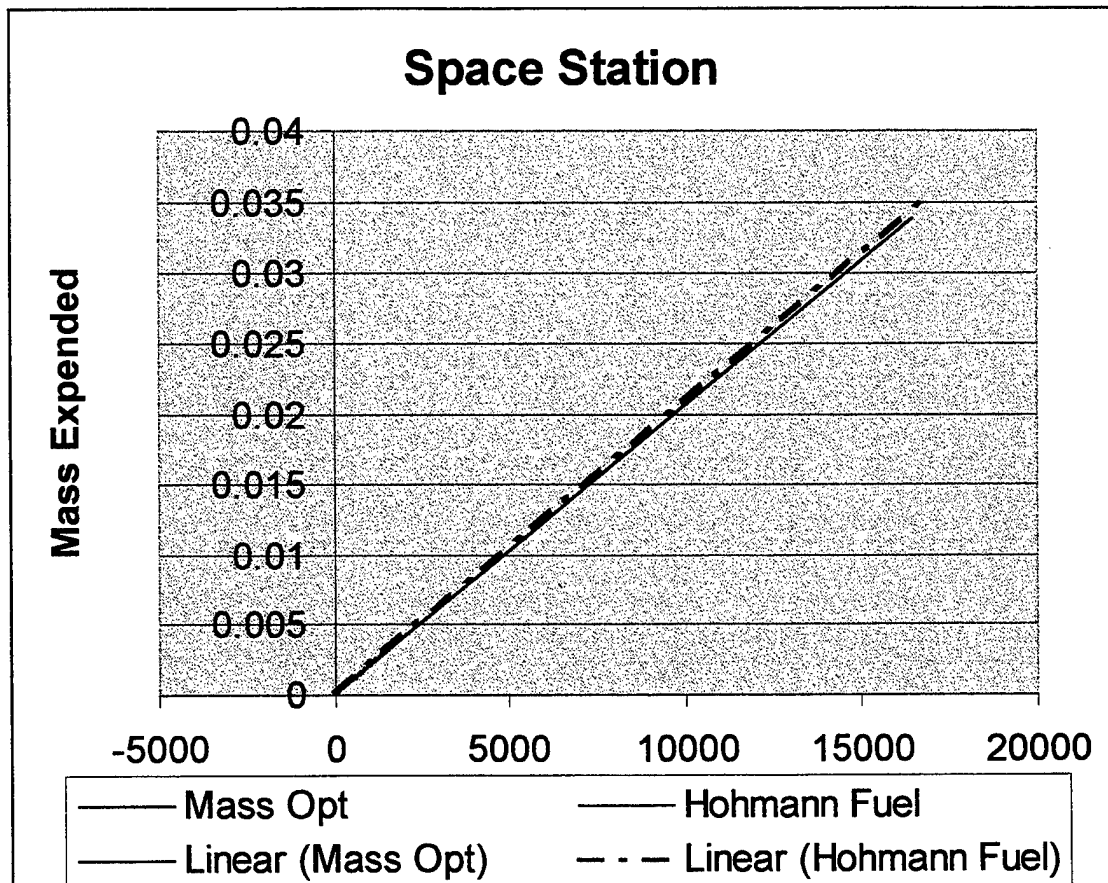


Figure 6-18

Figure 6-18 shows that again the optimal transfer is a more efficient maneuver than an impulsive Hohmann transfer. For the maximum thruster, of 10 units, drag losses would increase the consumption of propellant by the Hohmann transfer by over 10%. In terms of actual propellant, a sample at the 50-day mark would show that the optimal trajectory used 4252.9 kg of fuel for orbit maintenance while the Hohmann transfer would use 4337.2 kg. The savings in this case are approximately 2% over the ideal Hohmann, drag loss may increase that number to 13%. The optimal code would save about 600 kg per

year over the ideal Hohmann transfer, a savings of 6.6 million dollar per year in launch weight costs alone.

This example shows the usefulness of the code as well as the ease with which different spacecraft can be modeled. While a lower drag vehicle like the ISS has slightly less savings over the ideal Hohmann than a higher drag vehicle, the savings are present and significant over a long period of time. Further analysis is necessary to vary the parameters and understand the “physics” of the optimal trajectory.

## VII. CONCLUSION

The work succeeded in accomplishing the stated goals. Quantitatively, the rule of thumb is that the optimal trajectory is slightly better than the impulsive Hohmann (~ 3%) and significantly better than the finite-burn Hohmann (up to ~ 10%). The savings are at least equal to drag loss "elimination." Beyond the physical numbers, the crucial part of the thesis was to explore the orbit maintenance problem in depth and use optimal periodic control (OPC) theory as a tool to locate the fuel-optimal solution. The many simulations that were described were a very small portion of the learning and experimentation that went into attacking this process. The work began with general optimal control theory, later specialized by periodicity, the equations of motion for two-dimensional orbital travel, and a good numerical method. Examining the results of the preliminary simulations reveal some numerical instabilities in the optimization code which led to further examination of the equations in order to improve the reproducibility and accuracy of the results. With this accomplished, the optimal results could then be compared with various other trajectories in order to measure the optimal trajectory's fuel efficiency. The different controls and boundary conditions were modified in order to search for the most fuel-optimal trajectory.

The amount of information from these simulations is enormous. The optimal trajectory would represent a large departure from current orbit maintenance procedures. The first difference would manifest itself in the design process. Mission planners determine the periodicity of the orbit maintenance routine by either fixing an orbital band in which the satellite operates or by specifying specific times the spacecraft is to be at the lower altitude limit. With this information, the operating altitude, and the satellite's physical characteristics the optimal code can be run with varying thrust levels. By examining the control histories a maximum thrust can be determined. This is an absolute limit for the given conditions. Excess performance in the engine would not increase fuel efficiency. This limit gives designers an upper limit on the size of the needed thruster, creating possible savings of platform cost and weight.

Then there is the trajectory itself. A true departure from the Hohmann transfer, the optimal code calls for a smooth continuous throttle burn that transfers the vehicle to the top of the orbital band. The only requirement for the burn would be that since the burn

occurs over a larger period of time, it should be performed as an autonomous maneuver or be continually linked to the control of a ground station. Most current engines are normally bang-bang only. This trajectory gives enough justification by its savings to develop throttlable motors or develop chattering techniques which can model bang-bang engines as a varying thrust engine.

But with all good research there are new questions that can be addressed and studied. First the complexity of the atmospheric model can be increased. Diurnal effects may be able to increase the savings of trajectories. One of the strengths of this method is that modifying the atmospheric model only requires modification of the density function that is independent from the optimization code. Another area of future research is identifying more robust optimization codes and routines. A good portion of this work was spent in determining ways in which to increase the numerical accuracy of the results while also reducing the computer run time. New methods for numerical approximations may be highly productive and more accurate interpolation schemes may have a large benefit. The code is not limited to Earth bound operations. Simply by changing the constants throughout the programs and the density model, this method could be applied to any other planetary body.

The biggest strength of the program is its flexibility and applicability to different models. The normalization method made it very easy to modify the spacecraft characteristics or the size of the thrusters. In terms of physical characteristics, spacecraft are modeled by their mass to cross-sectional area ratios. This is what makes the ISS the same as the Hubble Space Telescope but different from an inflatable. The ability to run the code for different situations is an important aspect of the project's success.

Optimal control has long broken preconceptions about certain topics. The problem of orbit maintenance may be the latest example. In every primary orbital mechanics textbook there is a section on the Hohmann transfer that states that it is the most efficient transfer between circular orbits, which it is for many cases. But when complexities are added to the system, such as the addition of atmospheric drag, these statements do not hold true. It is the goal of every research project to question all the answers and push the boundaries of learning and understanding out a little further.

## LIST OF REFERENCES

1. Hernandez, A.A., *Single and Dual Burn Maneuvers for Low-Earth-Orbit Maintenance*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1994.
2. Gardner, P. A., *An Analysis of a Single-Burn Algorithm for Low-Earth Orbit Maintenance*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1994.
3. Ross, I. M., "Extremally Steered Singular Thrust-Arcs for Space Flight in an Atmosphere," *Acta Astronautica*, Vol. 39, No. 8, October 1996, pp. 569-577.
4. Ross, I. M., and Alfriend, K. T., "Low-Earth-Orbit Maintenance: Reboost vs Thrust- Drag Cancellation," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 4, Nov. 1994, pp. 930-932.
5. Elnagar, J., Kazemi, M.A. and Razzaghi, M., "The Pseudospectral Legendre Method for Discretizing Optimal Control Problem," *IEEE Transactions on Automatic Control*, Vol. 40, No. 10, 1995, pp. 1793-1796.
6. Elnagar, J. and Razzaghi, M. "A Collocation-Type Method for Linear Quadratic Optimal Control Problems," *Optimal Control Applications and Methods*, Vol. 18, 1997, pp. 227-235.
7. Fahroo, F., and Ross, I.M., "Costate Estimation by a Legendre Pseudospectral Method," to be presented in 1998 AIAA Guidance, Navigation and Control Conference, in Boston, MA, August 1998.
8. Larson, W.J., and Wertz, J. R. (eds.), *Space Mission Analysis and Design*, 2<sup>nd</sup> ed., Microcosm, Inc., Torrance, California, 1992.





## APPENDICES

### Appendix A - Density.m

```
% Program density.m
% Karl Jensen - May 13,1998

function rho = density(x);
rt = x * (6378+300);
    if rt >= 6753
        rref = 6778; h=58.2;
        rhoref = 2.62e-12;
    elseif rt >= 6703
        rref = 6728; h =54.8;
        rhoref = 6.66e-12;
    elseif rt >= 6653
        rref = 6678; h=50.3;
        rhoref = 1.87e-11;
    elseif rt >= 6603
        rref = 6628; h =44.8;
        rhoref = 5.97e-11;
    else rref = 6578; h= 37.5;
        rhoref = 2.41e-10;
    end;
rhon = rhoref*exp(-1*(rt-rref)/h);
rho = rhon/1.87e-11;
```

### Appendix B - Orbprop.m (First Order Propagator)

```
% First Order Propagator
% Karl Jensen - May 1, 1998

function ddot = orbprop(t,x)

global T alpha tau n band xt
global xt
% First Order Differential Equation Solver

% States
%          x(1) = r
%          x(2) = v
%          x(3) = gamma
%          x(4) = mass
%          x(5) = theta
m0 = 408420; a=2200;
% m0 = 3000; a =1000;
```

```

m0=3000;a=500;
isp = 300; gn = 9.81; cd = 2.35;
rok = 6378+300; rho0 = 1.87*10^(-11); rho =1;
v0 = sqrt((3.986005*10^5)/rok)*1000; r0 = rok * 1000;

t
B = 2 * m0 / (r0*rho0*cd*a);
ve = isp*gn/v0;

rho = density(x(1));
g = 1/(x(1)^2);
D = rho*(x(2)^2);
if (n>5)
    % I assume lower limit of the run = 0;
    taut = (2*t-tau)/tau;
    t
    Tp = interp1(xt,T,taut,'spline');
    if Tp < 0
        Tp = 0;
    end;
    alphap = interp1(xt,alpha,taut,'spline');
end;
if (n==1)
    Tp = D; alphap = 0;
end;
if (n==2)
    Tp = 0; alphap = 0;
end;
if (n==3)
    Tp = 20; alphap = 0;
end;

As = (Tp*cos(alphap)-D)/(x(4)*B);
An = Tp*sin(alphap)/(x(4)*B);

ddot(1) = x(2)*sin(x(3));
ddot(2) = -g*sin(x(3))+As;
ddot(3) = (x(2)^2/x(1)-g)*(cos(x(3))/x(2))+An/x(2);
ddot(4) = -abs(Tp)/(ve*B);
ddot(5) = x(2)*(cos(x(3))/x(1));
% 5 Equations of Motion

```

## APPENDIX C - HOH.M (HOHMANN TRANSFERS)

% Drag Code - Karl Jensen - Orbital Mechanics November 26,1996

global T

global band n tau ve

mu =1;

%band = 1.000147536-1;

m0 = 408420; a=2200;

%m0 = 3000; a =500;

%m0=3000;a=10000;

isp = 300; gn = 9.81; cd = 2.35;

rok = 6378+300; rho0 = 1.87\*10<sup>(-11)</sup>; rho =1;radref =rok;

v0 = sqrt((3.986005\*10<sup>5</sup>)/rok)\*1000; r0 = rok \* 1000;

ve = isp\*gn/v0;

% t,x are for Constant Thrust - Lower FKT

% tt, xx are for Free Fall

% tl, xl are mid band FKT values

r=1;

v=1;

gm=0;

m=1;

theta = 0;

p = 2\*pi\*(r<sup>3</sup>/mu)<sup>0.5</sup>;

tp = tau/(2\*pi)\*p;

% Constant Thrust

xo=[r v gm m theta];

to = 0;

tf = tp;

tol = 1.e-9;

n = 1

[t,x] = ode45('orbprop',to,tf,xo,tol,0);

figure(1);

orient tall;

subplot(2,2,1),plot(t/p,x(:,1),'b');xlabel('Orbits');title('Radius versus Orbit');ylabel('Radius (m)');

subplot(2,2,2),plot(t/p,x(:,2),'b');xlabel('Orbits');title('Velocity versus Orbit');ylabel('Velocity (m/s)');

```

subplot(2,2,3),plot(tt/p,xx(:,3)*360/(2*pi),'b');xlabel('Orbits');title('Gamma versus Orbit');
ylabel('gamma (deg)');
subplot(2,2,4),plot(tt/p,xx(:,4),'b');xlabel('Orbits');title('Mass versus Orbit');ylabel('Mass
(kg)');

% Free Fall
n = 2
sp = 5*p;
[tt,xx] = ode45('orbprop',to,sp,xo,tol,0);

temp = xx;temp2 = tt;

figure(2);
orient tall;
subplot(2,2,1),plot(tt/p,xx(:,1),'b');xlabel('Orbits');title('Radius versus Orbit');ylabel('Radius
(m)');
subplot(2,2,2),plot(tt/p,xx(:,2),'b');xlabel('Orbits');title('Velocity versus Orbit');
ylabel('Velocity (m/s)');
subplot(2,2,3),plot(tt/p,xx(:,3)*360/(2*pi),'b');xlabel('Orbits');title('Gamma versus Orbit');
ylabel('gamma (deg)');
subplot(2,2,4),plot(tt/p,xx(:,4),'b');xlabel('Orbits');title('Mass versus Orbit');ylabel('Mass
(kg)');

% Middle Band FKT
n = 1;
mid = r + band/2; lv = (mu/mid)^.5;
xo = [mid lv gm m theta]
[tl,xl] = ode45('orbprop',to,tf,xo,tol,0);

% High Band FKT
n=1;
higher = r + band; hv = (mu/higher)^.5;
xo = [higher hv gm m theta]
[th,xh] = ode45('orbprop',to,tf,xo,tol,0);

%Part Three - Hohmann Runs

flag = 1; n = 2;c=1;

while flag
    k = c;
    while (xx(k,1)>(r)) & flag
        k = k+1;
        if k >= size(tt,1)

```

```

        flag = 0;
    end;
end;
end;
if (c ~= 1) & flag
    burnvec = xx(k-1:k,:); burnt = tt(k-1:k);
    int = interp1(burnvec(:,1),burnt,1);
    invec = interp1(burnvec(:,1),burnvec,1);
    xx(k,:) = delvec(invec); tt(k) = int;

end;
if flag
    dr = pi*(cd*a/(m0*xx(k,4)))*rho0*(radref*1000*xx(k,1))^2;
    drn = dr / (radref*1000);
    atx = (1+r+band+0)/2;
    vfa = (1/xx(k,1))^0.5;
    vtxa = (2/xx(k,1)-1/atx)^0.5;
    dv = abs(vtxa-vfa); vn = xx(k,2)+dv;
    mf = xx(k,4)*exp(-dv/ve);
    n=2; txorb = [xx(k,1) vn xx(k,3) mf theta];
    tf = tt(k) + pi*(atx^3)^0.5;
    [tx,xp] = ode45('orbprop',tt(k),tf,txorb,tol,0);
    z = size(tx,1);
    vtxb = (((2/(r+band))-(1/atx)))^0.5;
    vfb = (1/(r+band))^0.5;
    dvb = abs(vfb-vtxb);
    mf = xp(z,4)*exp(-dvb/ve);
    vn = dvb + xp(z,2);
    to = tf; tf = tp;
    orb = [xp(z,1) vn xp(z,3) mf xp(z,5)];
    ovr = 6*(band+0.005)/(drn);
    if (tf-to) > ovr
        tf = to + ovr;
    end;
    [tq,xq] = ode45('orbprop',to,tf,orb,tol,0);
    ff = [5 1];
    if ff == size(xq)
        xq=xq';
    end;
    st = [tt(1:k); tx; tq];
    sx = [xx(1:k,:); xp; xq];
    tt = st;
    xx = sx;
    c = k+z;
end;
end
end

```

```

figure(3);
xx=temp;tt=temp2;
orient tall;
subplot(2,2,1),plot(st/p,sx(:,1),'b');
xlabel('Orbits');title('Radius versus Orbit');ylabel('Radius (m)');
subplot(2,2,2),plot(st/p,sx(:,2),'b');xlabel('Orbits');
title('Velocity versus Orbit');ylabel('Velocity (m/s)');
subplot(2,2,3),plot(st/p,sx(:,3)*360/(2*pi),'b');
xlabel('Orbits');title('Gamma versus Orbit');ylabel('gamma (deg)');
subplot(2,2,4),plot(st/p,sx(:,4),'b');xlabel('Orbits');
title('Mass versus Orbit');ylabel('Mass (kg)');
sx1 =sx;st1=st;

```

```

figure(4);
orient tall;
subplot(2,2,1),plot(st/p,sx(:,1),'b');
xlabel('Orbits');title('Radius versus Orbit');ylabel('Radius (m)');
subplot(2,2,2),plot(st/p,sx(:,2),'b');xlabel('Orbits');
title('Velocity versus Orbit');ylabel('Velocity (m/s)');
subplot(2,2,3),plot(st/p,sx(:,3)*360/(2*pi),'b');
xlabel('Orbits');title('Gamma versus Orbit');ylabel('gamma (deg)');
subplot(2,2,4),plot(st/p,sx(:,4),'b');xlabel('Orbits');
title('Mass versus Orbit');ylabel('Mass (kg)');

```

```

figure(5)
plot(st1/p,m-sx1(:,4),'b',t/p,m-x(:,4),'m--',tl/p,m-xl(:,4),'m:',th/p,m-xh(:,4),'m-');title('Propellant Comparison');
xlabel('Orbits');ylabel('Normalized Mass');
hold;
%plot(st1/p,m-sx1(:,4),'r');
hold off;
k=size(xl(:,4)); mid = m-xl(k,4)
k=size(xh(:,4)); high = m-xh(k,4)
k=size(x(:,4)); low = m-x(k,4)

hohfuel = m-sx1(:,4);lowfuel = m-x(:,4);
midfuel = m-xl(:,4);highfuel = m-xh(:,4);
hohtime = st1;

```

## Appendix D - Orbopt.m

```
% This routine calculates the minimum of the cost function given by  
% the function crit.
```

```
clear;  
tic;
```

```
global n  
global Dn x w  
global B ve rho aop aref
```

```
n=24; % The number of lobatto points.
```

```
%m0 = 408420; a=2200;  
m0 = 3000; a =500;  
%m0=3000;a=10000;  
isp = 300; gn = 9.81; cd = 2.35;  
rok = 6378+300; rho0 = 1.87*10^(-11); rho =1;  
v0 = sqrt((3.986005*10^5)/rok)*1000; r0 = rok * 1000;
```

```
B = 2 * m0 / (r0*rho0*a*cd);
```

```
ve = isp*gn/v0;
```

```
Tguess = 5.0;  
[Dn,x,w]=diffm(n);
```

```
aop(1:n)= ones(n,1);  
aop(n+1:2*n) = ones(n,1);  
aop(2*n+1:3*n) = 1.1*ones(n,1);  
aop(3*n+1:4*n) = ones(n,1);  
aop(4*n+1:5*n) = Tguess*ones(n,1);  
aop(5*n+1:6*n) = 0*ones(n,1);  
aop(6*n+1:7*n) = zeros(n,1);  
aop(7*n+1) =112;  
taup=aop(7*n+1);  
options(13)=5*n+6;  
options(14) = 400*(7*n+1);  
options(4)=10^(-4);%5  
options(3)=10^(-5);%5  
options(2)=10^(-5);%5  
options(1)=1;  
%options(18)=1;  
options(16) = 10^(-6);%7  
vub=[ ];
```

```

vlb=[ ];

aop=constr('dorbcrit',aop,options,vlb,vub);

% a is for the state and b is for the control.
rp=aop(1:n);
vp = aop(n+1:2*n);
gammap = aop(2*n+1:3*n);
massp = aop(3*n+1:4*n);
Tp = aop(4*n+1:5*n);
alphap = aop(5*n+1:6*n);
thetap = aop(6*n+1:7*n);
tau = aop(7*n+1);

toc;

tic;
orbres;
orbconv;

toc;

```

## Appendix E - Orbcrit.m

```

function [costfn,g]=dorbcrit(aop);

%% This function calculates the cost function that is to be minimized
%% and the state constraints.

global n x w Dn;
global B ve rho aref;
% global costfn;

% Set up the cost function.
for i=1:n
    fn(i)=aop(4*n+i);
end;
costfn= 1/(2)*sum(w.*fn');
tau = aop(7*n+1);

% Set up the state constraints.
for i=1:n
    rt = aop(i)*(6378+300);
    if rt >= 6728

```



```

        rref = 6778; h=58.2;
        rhoref = 2.62e-12;
    elseif rt >= 6628
        rref = 6678; h =50.3;
        rhoref = 1.87e-11;
    else rref = 6528; h= 37.5;
        rhoref = 2.41e-10;
    end;
rhon = rhoref*exp(-1*(rt-rref)/h);
rho = rhon/1.87e-11;

G(i) = 1/(aop(i)^2);
D(i) = rho*(aop(n+i)^2);
Ast = (aop(4*n+i)*cos(aop(5*n+i))-D(i))/(aop(3*n+i)*B);
Ant = aop(4*n+i)*sin(aop(5*n+i))/(aop(3*n+i)*B);
V(i) = ve;

% Theta
g(i+4*n) = (2/tau)*sum(Dn(i,:). *aop(6*n+1:7*n))-(aop(n+i)/aop(i)*cos(aop(2*n+i)));
% Mass
g(i+3*n) = (2/tau)*sum(Dn(i,:). *aop(3*n+1:4*n))+(aop(4*n+i)/(ve*B));
% Gamma
g(i+2*n) = (2/tau)*sum(Dn(i,:). *aop(2*n+1:3*n));
g(i+2*n) = g(i+2*n)-(((aop(n+i)^2/aop(i))-G(i))*(cos(aop(2*n+i))/aop(i+n)));
g(i+2*n) = g(i+2*n)- Ant/aop(n+i);

% Radius
g(i) = (2/tau)*sum(Dn(i,:). *aop(1:n))-aop(n+i)*sin(aop(2*n+i));
% Velocity
g(i+n) = (2/tau)*sum(Dn(i,:). *aop(n+1:2*n))+(G(i)*sin(aop(2*n+i)))-Ast;

g(i+5*n+6) = - aop(4*n+i);
g(8*n+6+i) = aop(4*n+i)-5.0;
% g(i+9*n+6) = 1-aop(i);
%g(i+7*n+5) = aop(i)-1;
g(i+6*n+6) = aop(5*n+i)-pi;
g(i+7*n+6) = -pi-aop(5*n+i);
end;

g(5*n+1) = aop(1)-aop(n);
g(5*n+2) = aop(2*n+1)-aop(3*n);
g(5*n+3) = aop(6*n+1);
g(5*n+4) = aop(n+1)-aop(2*n);
g(5*n+5) = aop(3*n+1)-1;

```

```
g(5*n+6) = aop(1)-1;
```

```
g=g';
```

## Appendix F - Orbconv.m

```
% Conversion from AOPs to Functions of Time
```

```
global aop n  
global T alpha tau  
global band xt
```

```
r0 = aop(1); v0 = aop(n+1); gm0 = aop(2*n+1);  
m0 = aop(3*n+1); th0 = aop(6*n+1);
```

```
T = aop(4*n+1:5*n);  
alpha = aop(5*n+1:6*n);
```

```
tau = aop(7*n+1);  
xt=x;
```

```
xo=[r0 v0 gm0 m0 th0];  
to = 0;  
tf = tau;  
tol = 1.e-9;
```

```
[t,x] = ode45('orbprop',to,tf,xo,tol,1);
```

```
n1 = size(t);  
rr(1) = aop(1); rr(n1(1,1)) = aop(n);  
vr(1) = aop(1+n); vr(n1(1,1)) = aop(2*n);  
gmr(1) = aop(1+n*2); gmr(n1(1,1)) = aop(3*n);  
massr(1) = aop(1+3*n); massr(n1(1,1)) = aop(4*n);  
thrustr(1) = aop(1+n*4); thrustr(n1(1,1)) = aop(5*n);  
alphar(1) = aop(1+5*n); alphar(n1(1,1)) = aop(6*n);  
thetar(1) = aop(1+6*n); thetar(n1(1,1)) = aop(7*n);  
for k = 2:(n1-1)  
    taut = (2*t(k)-tau)/tau;  
k  
rr(k) = interp1(xt,aop(1:n),taut,'spline');  
vr(k) = interp1(xt,aop(n+1:2*n),taut,'spline');  
gmr(k) = interp1(xt,aop(2*n+1:3*n),taut,'spline');  
massr(k) = interp1(xt,aop(3*n+1:4*n),taut,'spline');
```

```

    if massr(k) > massr(k-1)
        massr(k)=massr(k-1);
    end;
thrustr(k) = interp1(xt,aop(4*n+1:5*n),taut,'spline');
    if thrustr(k) < 0
        thrustr(k) = 0;
    end;

alphar(k) = interp1(xt,alpha,taut,'spline');
thetar(k) = interp1(xt,aop(6*n+1:7*n),taut,'spline');

end;

figure(1)
erad = x(:,1)'-rr; evel = x(:,2)'-vr; emass = x(:,4)'-massr;
subplot(2,1,1),plot(t,(x(:,1)'-rr),'b',t,(x(:,2)'-vr),'r');
title('Radius Error (Blue) Velocity Error (red)');
subplot(2,1,2),plot(t,(x(:,3)'-gmr),'b',t,(x(:,4)'-massr),'r');
title('Gamma Error(Blue) Mass Error(Red)');

figure(2)

subplot(2,2,1),plot(t,rr,'r');title('Radius vs Time');
subplot(2,2,2),plot(t,vr,'r');title('Velocity vs Time');
subplot(2,2,3),plot(t,gmr,'r');title('Gamma vs Time');
subplot(2,2,4),plot(t,massr,'r');title('Mass vs Time');

figure(3)
subplot(2,1,1),plot(t,thrustr,'r');title('Thrust vs Time');
subplot(2,1,2),plot(t,alphar,'r');title('Alpha vs Time');

band = max(rr)-min(rr);
rprop = x(:,1); vprop =x(:,2); gmprop = x(:,3);mprop =x(:,4);
thprop = x(:,5);

```

## Appendix G - Orbres.m

```

global aop costfn
[costfn,g]=dorbcrit(aop)

costfn

r=aop(1:n)
v = aop(n+1:2*n)

```

```

gamma = aop(2*n+1:3*n)
mass = aop(3*n+1:4*n)
T = aop(4*n+1:5*n)
alpha = aop(5*n+1:6*n)
theta = aop(6*n+1:7*n)
tau = aop(7*n+1)
y = (tau/2)*(x+1);

```

## Appendix H - Lobatto.m

```
% here are the files for lobatto points.
```

```
% [x w]=lobatto(n) n= number of points.
```

```
function [x,w] = lobatto(n,a,b)
```

```
% [x w] = lobatto(n) or [x w] = lobatto(n,alpha,beta):
```

```
%
```

```
% Computes abscissa and weights for the n-point Gauss-Jacobi-Lobatto
% quadrature formula using the method of Gene H. Golub, Some modified
% matrix eigenvalue problems, SIAM Rev. 15 (1973) 318-334. Another early
% algorithm for this is by David Galant, An implementation of Christoffel's
% formula in the theory of orthogonal polynomials, Math. Comp. 25 (1971)
% 111-113. All such algorithms should be "reviewed", in light of recent
% improvements in tqr and Cholesky LR algorithms. But, this algorithm
% "ain't bad".
```

```
% Copyright (c) 23 August 1997 by Bill Gragg. All rights reserved.
```

```
% lobatto calls mxt, mxtj and tqr.
```

```
% begin lobatto
```

```
if nargin < 2
```

```
    a = 0; b = 0;
```

```
end
```

```
m = 2^(a + b + 1)*beta(a+1,b+1); us = a == b;
```

```
n = n - 1; [a b] = mxtj(n,a,b); T = mxt(a,b);
```

```
I = eye(n); e = zeros(n,1); e(n) = 1;
```

```
c = (T + I)\e; c = c(n); d = (T - I)\e;
```

```
d = d(n); e = c - d; c = (c + d)/e;
```

```
d = sqrt(2/e); a(n+1) = c; b(n) = d;
```

```
[x u] = tqr(a,b); u = u'; w = m*u.^2;
```

```

% "Purify" formulas in the ultraspherical case.
if us
    x = (x - flipud(x))/2; w = (w + flipud(w))/2;
end

% end lobatto

```

## Appendix I - Diffm.m

```

function [Dn,x,w]=Diff(n);
% This function calculates the differentiation matrix Dn that is
% obtained by differentiating the Legendre Polynomials at the legendre-
% Lobatto points. It's zero on the main diagonal except at l=k=1, where
% Dn(1,1)= n(n+1)/4; and at l=k=n; where Dn(n,n)=-n(n+1)/4.
% n= no of Lobatto points. For the other points l (~)k, we have
% Dn(l,k)= Ln(xl)/Ln(xk)*(1/xl-xk).

```

```

[x w]=lobatto(n);
x=sort(x);
% initialize Dn
Dn=zeros(n);
Dn(1,1)=-(n-1)*n/4;
Dn(n,n)=n*(n-1)/4;

% Calculate the legendre polynomials at xi.
p=0*eye(n);
for i=1:n; s=x(i); p(i,1)=1; p(i,2)=s;
for j=2:n-1; p(i,j+1)=((2*j-1)*s*p(i,j)-(j-1)*p(i,j-1))/j; end; end;

% Fill out the rest of matrix Dn.
for l=1:n; for k=1:n;
if l~=k,
Dn(l,k)=p(l,n)/(p(k,n)*(x(l)-x(k)));
end;
end;end;

```

## Appendix J - Tfopt.m

```

% This routine calculates the minimum of the cost function given by
% the function crit.
clear;
tic;

```

```

global n
global Dn x w
global B ve rho aop aref

n=24; % The number of lobatto points.

%m0 = 408420; a=2200;
m0 = 3000; a =500;
%m0=3000;a=10000;
isp = 300; gn = 9.81; cd = 2.35;
rok = 6378+300; rho0 = 1.87*10^(-11); rho =1;
v0 = sqrt((3.986005*10^5)/rok)*1000; r0 = rok * 1000;

B = 2 * m0 / (r0*rho0*a*cd);

ve = isp*gn/v0;

Tguess = 5.0;
[Dn,x,w]=diffm(n);

%% the true solutions. u is the control and z is the state variable.
y=(x+1)/2;

aop(1:n)= ones(n,1);
aop(n+1:2*n) = ones(n,1);
aop(2*n+1:3*n) = 1.1*ones(n,1);
aop(3*n+1:4*n) = ones(n,1);
aop(4*n+1:5*n) = Tguess*ones(n,1);
aop(5*n+1:6*n) = 0*ones(n,1);
aop(6*n+1:7*n) = zeros(n,1);
aop(7*n+1) =112;
taup=aop(7*n+1);
options(13)=5*n+7;
options(14) = 400*(7*n+1);
options(4)=10^(-4);%5
options(3)=10^(-5);%5
options(2)=10^(-5);%5
options(1)=1;
%options(18)=1;
options(16) = 10^(-6);%7
vub=[ ];
vlb=[ ];

```

```

aop=constr('tfcrit',aop,options,vlb,vub);

% a is for the state and b is for the control.
rp=aop(1:n);
vp = aop(n+1:2*n);
gammap = aop(2*n+1:3*n);
massp = aop(3*n+1:4*n);
Tp = aop(4*n+1:5*n);
alphap = aop(5*n+1:6*n);
thetap = aop(6*n+1:7*n);
tau = aop(7*n+1);

toc;

tic;
orbres;
orbconv;

```

## Appendix K -Tfcrit.m

```

function [costfn,g]=tfcrit(aop);

%% This function calculates the cost function that is to be minimized
%% and the state constraints.

global n x w Dn;
global B ve rho aref;
% global costfn;

% Set up the cost function.
for i=1:n
    fn(i)=aop(4*n+i);
end;
costfn= 1/(2)*sum(w.*fn);
tau = aop(7*n+1);

% Set up the state constraints.
for i=1:n
    rt = aop(i)*(6378+300);
    if rt >= 6728
        rref = 6778; h=58.2;
        rhoref = 2.62e-12;
    elseif rt >= 6628
        rref = 6678; h =50.3;
    end
end

```

```

        rhoref = 1.87e-11;
    else rref = 6528; h= 37.5;
        rhoref = 2.41e-10;
    end;
    rhon = rhoref*exp(-1*(rt-rref)/h);
    rho = rhon/1.87e-11;

    G(i) = 1/(aop(i)^2);
    D(i) = rho*(aop(n+i)^2);
    Ast = (aop(4*n+i)*cos(aop(5*n+i))-D(i))/(aop(3*n+i)*B);
    Ant = aop(4*n+i)*sin(aop(5*n+i))/(aop(3*n+i)*B);
    V(i) = ve;

% Theta
g(i+4*n) = (2/tau)*sum(Dn(i,:).*aop(6*n+1:7*n))-aop(n+i)/aop(i)*cos(aop(2*n+i));
% Mass
g(i+3*n) = (2/tau)*sum(Dn(i,:).*aop(3*n+1:4*n))+aop(4*n+i)/(ve*B);
% Gamma
g(i+2*n) = (2/tau)*sum(Dn(i,:).*aop(2*n+1:3*n));
g(i+2*n) = g(i+2*n)-(((aop(n+i)^2)/aop(i))-G(i))*(cos(aop(2*n+i))/aop(i+n));
g(i+2*n) = g(i+2*n)- Ant/aop(n+i);

% Radius
g(i) = (2/tau)*sum(Dn(i,:).*aop(1:n))-aop(n+i)*sin(aop(2*n+i));
% Velocity
g(i+n) = (2/tau)*sum(Dn(i,:).*aop(n+1:2*n))+G(i)*sin(aop(2*n+i))-Ast;

g(i+5*n+7) = - aop(4*n+i);
g(8*n+7+i) = aop(4*n+i)-5.0;
% g(i+9*n+7) = 1-aop(i);
%g(i+7*n+7) = aop(i)-1;
g(i+6*n+7) = aop(5*n+i)-pi;
g(i+7*n+7) = -pi-aop(5*n+i);
end;

g(5*n+1) = aop(1)-aop(n);
g(5*n+2) = aop(2*n+1)-aop(3*n);
g(5*n+3) = aop(6*n+1);
g(5*n+4) = aop(n+1)-aop(2*n);
g(5*n+5) = aop(3*n+1)-1;
g(5*n+6) = aop(1)-1;
g(5*n+7) = aop(7*n+1)-112;

g=g';

```



## Appendix L - Notopt.m

```
% This routine calculates the minimum of the cost function given by  
% the function crit.
```

```
clear;  
tic;
```

```
global n  
global Dn x w  
global B ve rho aop aref
```

```
n=24; % The number of lobatto points.
```

```
%m0 = 408420; a=2200;  
m0 = 3000; a =500;  
%m0=3000;a=10000;  
isp = 300; gn = 9.81; cd = 2.35;  
rok = 6378+300; rho0 = 1.87*10(-11); rho =1;  
v0 = sqrt((3.986005*10(5))/rok)*1000; r0 = rok * 1000;
```

```
B = 2 * m0 / (r0*rho0*a*cd);
```

```
ve = isp*gn/v0;
```

```
Tguess = 25.0;  
[Dn,x,w]=diffm(n);
```

```
aop(1:n)= ones(n,1);  
aop(n+1:2*n) = ones(n,1);  
aop(2*n+1:3*n) = 1.1*ones(n,1);  
aop(3*n+1:4*n) = ones(n,1);  
aop(4*n+1:5*n) = Tguess*ones(n,1);  
aop(5*n+1:6*n) = 0*ones(n,1);
```

```
taup=112.6;  
options(13)=4*n+5;  
options(14) = 400*(7*n+1);  
options(4)=10(-6);%5  
options(3)=10(-8);%5  
options(2)=10(-8);%5  
options(1)=1;
```

```

%options(18)=1;
options(16) = 10^(-7);%7
vub=[ ];
vlb=[ ];

aop=constr('notcrit',aop,options,vlb,vub);

% a is for the state and b is for the control.
rp=aop(1:n);
vp = aop(n+1:2*n);
gammap = aop(2*n+1:3*n);
massp = aop(3*n+1:4*n);
Tp = aop(4*n+1:5*n);
alphap = aop(5*n+1:6*n);

tau = 112.6;

toc;

tic;
notres;
notconv;

toc;

```

## Appendix M - Notcrit.m

```

function [costfn,g]=notcrit(aop);

%% This function calculates the cost function that is to be minimized
%% and the state constraints.

global n x w Dn;
global B ve rho aref;
% global costfn;

% Set up the cost function.
for i=1:n
    fn(i)=aop(4*n+i);
end;
costfn= 1/(2)*sum(w.*fn');
tau = 112.6;

% Set up the state constraints.

```

```

for i=1:n
rho = density(aop(i));

G(i) = 1/(aop(i)^2);
D(i) = rho*(aop(n+i)^2);
Ast = (aop(4*n+i)*cos(aop(5*n+i))-D(i))/(aop(3*n+i)*B);
Ant = aop(4*n+i)*sin(aop(5*n+i))/(aop(3*n+i)*B);
V(i) = ve;

% Theta

% Mass
g(i+3*n) = (2/tau)*sum(Dn(i,:).*aop(3*n+1:4*n))+(aop(4*n+i)/(ve*B));
% Gamma
g(i+2*n) = (2/tau)*sum(Dn(i,:).*aop(2*n+1:3*n));
g(i+2*n) = g(i+2*n)-(((aop(n+i)^2)/aop(i))-G(i))*(cos(aop(2*n+i))/aop(i+n));
g(i+2*n) = g(i+2*n)- Ant/aop(n+i);

% Radius
g(i) = (2/tau)*sum(Dn(i,:).*aop(1:n))-aop(n+i)*sin(aop(2*n+i));
% Velocity
g(i+n) = (2/tau)*sum(Dn(i,:).*aop(n+1:2*n))+(G(i)*sin(aop(2*n+i)))-Ast;

g(i+4*n+5) = - aop(4*n+i);
g(5*n+5+i) = aop(4*n+i)-5.0;
g(i+6*n+5) = aop(5*n+i)-pi;
g(i+7*n+5) = -pi-aop(5*n+i);
end;

g(4*n+1) = aop(1)-aop(n);
g(4*n+2) = aop(2*n+1)-aop(3*n);
g(4*n+3) = aop(n+1)-aop(2*n);
g(4*n+4) = aop(3*n+1)-1;
g(4*n+5) = aop(1)-1;

g=g';

```

## Appendix N - Noaopt.m

```

% This routine calculates the minimum of the cost function given by
% the function crit.
clear;
tic;

```

```

global n rhom vm
global Dn x w
global B ve rho aop aref band
global tg ag taup

n=24; % The number of lobatto points.
band = 0.01;
rhom = density(1+band/2);
vm= sqrt(1/(1+band/2));

m0 = 408420; a=2200;
m0 = 3000; a =500;
%m0=3000;a=10000;
isp = 300; gn = 9.8067; cd = 2.35;
rok = 6378+300; rho0 = 1.87*10^(-11); rho =1;
v0 = sqrt((3.986005*10^5)/rok)*1000; r0 = rok * 1000;

B = 2 * m0 / (r0*rho0*a*cd);

ve = isp*gn/v0;

Tguess = 20;
[Dn,x,w]=diffm(n);

aop(1:n)= ones(n,1);
aop(n+1:2*n) = ones(n,1);
aop(2*n+1:3*n) = .5*ones(n,1);
aop(3*n+1:4*n) = ones(n,1);
aop(4*n+1:5*n) = Tguess*ones(n,1);
%aop(4*n+8:5*n) = ones(length(aop(4*n+8:5*n)),1);

taup=112.6;
options(13)=4*n+5;
options(14) = 400*(7*n+1);
options(4)=10^(-7);%5
options(3)=10^(-8);%5
options(2)=10^(-9);%5
options(1)=1;
%options(18)=1;
options(16) = 10^(-8);%7
vub=[ ];
vlb=[ ];

aop=constr('noacrit',aop,options,vlb,vub);

```

```
% a is for the state and b is for the control.
```

```
rp=aop(1:n);  
vp = aop(n+1:2*n);  
gammap = aop(2*n+1:3*n);  
massp = aop(3*n+1:4*n);  
Tp = aop(4*n+1:5*n);
```

```
tau = taup;
```

```
toc;
```

```
tic;  
noares;  
noaconv;
```

```
toc;
```

## Appendix O - Noacrit.m

```
function [costfn,g]=noacrit(aop);
```

```
%% This function calculates the cost function that is to be minimized  
%% and the state constraints.
```

```
global n x w Dn band;  
global B ve rho aref;  
global rhom vm taup;  
% global costfn;
```

```
% Set up the cost function.
```

```
i=1:n;  
% fn=aop(4*n+i)/(rhom*vm^2);  
fn=aop(4*n+i);  
costfn= 1/(2)*sum(w.*fn');  
tau = 112.6;
```

```
% Set up the state constraints.
```

```
for i=1:n
```

```
rho = density(aop(i));
```

```
G(i) = 1/(aop(i)^2);
```

```

D(i) = rho*(aop(n+i)^2);
Ast = (aop(4*n+i)*1-D(i))/(aop(3*n+i)*B);
Ant = aop(4*n+i)*0/(aop(3*n+i)*B);
V(i) = ve;

% Theta

% Mass
g(i+3*n) = (2/tau)*sum(Dn(i,:).*aop(3*n+1:4*n))+(aop(4*n+i)/(ve*B));
% Gamma
g(i+2*n) = (2/tau)*sum(Dn(i,:).*aop(2*n+1:3*n));
g(i+2*n) = g(i+2*n)-(((aop(n+i)^2)/aop(i))-G(i))*(cos(aop(2*n+i))/aop(i+n));
g(i+2*n) = g(i+2*n)- Ant/aop(n+i);

% Radius
g(i) = (2/tau)*sum(Dn(i,:).*aop(1:n))-aop(n+i)*sin(aop(2*n+i));
% Velocity
g(i+n) = (2/tau)*sum(Dn(i,:).*aop(n+1:2*n))+(G(i)*sin(aop(2*n+i)))-Ast;

g(i+4*n+5) = - aop(4*n+i);
g(5*n+5+i) = aop(4*n+i)-20.0;
g(i+6*n+5) = aop(i)-(1+band);

end;

g(4*n+1) = aop(1)-aop(n);
g(4*n+2) = aop(2*n+1)-aop(3*n);
g(4*n+3) = aop(n+1)-aop(2*n);
g(4*n+4) = aop(3*n+1)-1;
g(4*n+5) = aop(1)-1;

g=g';

```

## Appendix P - Tqr.m

```

% Total flops (scalar case, see csgn): TBC

% Problem.

```

```
% 1. Compare this function experimentally with csgn. Compare with regard
% to both execution time and numerical stability. Is matlab computing
% sign correctly?
```

```
function [lam,U] = tqr(a,b,U)
```

```
% [lam u] = tqr(a,b) or [lam U] = tqr(a,b,U):
```

```
%
```

```
% [lam u] = tqr(a,b):
```

```
%
```

```
% The column lam contains the eigenvalues of the Hermitian tridiagonal
% matrix  $T = \text{mxt}(a,b)$  computed by one version of the (real symmetric) tqr
% algorithm with Wilkinson's shift. The column u contains the first
% elements of the eigenvectors of T normalized to be nonnegative and such
% that the eigenvectors are unit vectors. In practice this is an  $O(n^2)$ 
% process. If u is omitted only the eigenvalues are computed. The
% computed eigenvalues are real and are sorted to be nonincreasing.
```

```
%
```

```
% [lam U] = tqr(a,b,U):
```

```
%
```

```
% This replaces the input U by UV with V a matrix of orthonormal eigen-
% vectors of T. If the input U is I the output U is V. If the input U is
% unitary with  $AU = UT$  then the output U is unitary with  $AU = UD$  and  $D =$ 
%  $\text{diag}(\text{lam})$ .
```

```
%
```

```
% If the input U is  $e(1)'$  the output U is  $u'$ . If the input U is
%  $[e(1)'; e(n)']$  the output U is  $[u'; v']$  with v the column of last
% elements of the normalized eigenvectors. If the subdiagonal elements of
% T are all nonzero then the elements of v alternate in sign, at least
% mathematically.
```

```
% Copyright (c) 2 February 1991 by Bill Gragg. All rights reserved.
```

```
% Revised 15 July 1994.
```

```
% tqr calls sgn.
```

```
% begin tqr
```

```
% Ensure that T is Hermitian and shift b down one unit.
```

```

a = real(a); n = length(a); b = [0; b(:)]; b = b(1:n);

% Initialize U if required and execute a diagonal unitary similarity
% transformation to make T have nonnegative subdiagonal elements.

if nargout > 1

    if nargin < 3
        U = zeros(1,n); U(1) = 1;
    end

    u = sgn(b); u = cumprod(u); U = U*diag(u);

end

b = abs(b);

% Scale the matrix up by a power of two to give nearly the widest
% possible exponent range.

scale = norm([a; b*sqrt(2)]); scale = 2^(1024 - ceil(log2(scale)));
a = a*scale; b = b*scale;

format compact % Temporary statements
maxscale = max(abs([a; b])); % for display.

% "Do tqr".

for m = n:-1:1

% disp(m) % Temporary display statement.

% Compute the mth eigenvalue.

for its = 0:10*n % its is the iteration index.

% Split the matrix if possible. This is also the termination
% test.

for k = m:-1:1

    if k > 1

        tol = abs(a(k-1)) + abs(a(k));

```



```

    if tol + b(k) == tol
        b(k) = 0; break
    end

end

end

if k == m

    break % b(m) = 0. a(m) is an eigenvalue.

else

    if its == 10*n
        error('tqr iteration did not terminate in 10n steps!')
    end

%     Compute Wilkinson's shift w as a perturbation of the
%     Rayleigh shift r = a(m). As the algorithm converges
%     c = b(m) --> 0.

r = a(m); c = b(m); d = (r - a(m-1))/2; s = abs(d);

if c < s
    s = c/s; t = 1 + sqrt(1 + s*s); t = c*s/t; % t < c;
else
    s = s/c; t = s + sqrt(1 + s*s); t = c/t; % t < c;
end

if d > 0
    w = r + t;
else
    w = r - t;
end

%     Take a step of the tqr algorithm. There are many ways to
%     implement the inner loop. We recently found the fastest
%     known stable form in terms of flops. The form given here
%     is elegant.

c = 1; s = 0; p = w - a(k); t = p;

for j = k:m-1

```

```

%           Compute the two by two reflector stably and update b(j).

oldc = c; oldt = t; q = b(j+1); u = abs(p);

if q < u
    v = q/u;    r = sqrt(1 + v*v); b(j) = u*r*s;
    u = sgn(p); c = u/r;          s = v/r;
else
    v = p/q;    r = sqrt(1 + v*v); b(j) = q*r*s;
    u = 1;      c = v/r;          s = u/r;
end

%           Update p, t, a(j), and U(:,j+1) if required.

p = c*(w - a(j+1)) - s*q*oldc; t = c*p;
a(j) = a(j+1) + t - oldt;

if nargout > 1
    i = j:j+1; U(:,i) = U(:,i)*[-c s; s c];
end

end

%           Update b(m), a(m), and U(:,m) if required.

b(m) = abs(p)*s; a(m) = w - t; c = sgn(p);

%           disp(b(m)/maxscale) % Temporary display statement.

if nargout > 1
    U(:,m) = - U(:,m)*c;
end

end

end

%           pause(3) % Temporary pause statement.

end

%           Sort and prepare the output.

[a p] = sort(-a); lam = - a/scale;

```

```

if nargout > 1

    U = U(:,p); u = U(1,:);

    if nargin < 3
        u = abs(u); U = u';
    else
        u = sgn(u); U = U*diag(u');
    end

end

% end tqr

```

## Appendix Q - Mxt.m

```
% Problems.
```

```
% 1. Relate  $T = \text{mxt}(a,b)$ , with  $[a \ b] = \text{mxtj}(n,1/2)$ , with the negative second
% difference matrix  $S = \text{mxt}(c,d)$ , with  $[c \ d] = \text{mxs}(n)$ .
```

```
function T = mxt(a,b,c)
```

```

% T = mxt(a,b,c) or T = mxt(a,b):
%
% T = mxt(a,b,c) is the TRIDIAGONAL MATRIX with diagonal elements a(1:n),
% subdiagonal elements b(1:n-1) and superdiagonal elements c(1:n-1).
%
% T = mxt(a,b) is the HERMITIAN tridiagonal matrix with diagonal elements
% real(a(1:n)) and subdiagonal elements b(1:n-1).
% Copyright (c) 1 December 1990 by Bill Gragg. All rights reserved.
% Revised 21 November 1992.
% mxt calls no extrinsic functions.
% begin mxt

```

```

if nargin < 3
    a = real(a); c = b';
end

n = length(a); b = b(1:n-1); c = c(1:n-1); z = zeros(n-1,1);

if n < 500

    B = diag(b); B = [z' 0; B z]; C = diag(c); C = [z C; 0 z'];

```

```

    T = diag(a); T = T + B + C;

else

    T = zeros(n);

    for k = 1:n-1
        T(k,k) = a(k); T(k+1,k) = b(k); T(k,k+1) = c(k);
    end

    T(n,n) = a(n);

end

% end mxt

```

## Appendix R - Mxtj.m

```

function [a,b] = mxtj(n,alpha,beta)

% [a b] = mxtj(n,alpha,beta), [a b] = mxtj(n,alpha), [a b] = mxtj(n),
% T = mxtj(n,alpha,beta), T = mxtj(n,alpha) or T = mxtj(n):
%
% mxtj(n,alpha,beta): T = mxt(a,b) is the Jacobi matrix whose characteristic
% polynomial p is (a nonzero scalar multiple of) the nth JACOBI polynomial.
% The eigenvalues of T are the abscissas of the nth order Gauss-Christoffel
% quadrature formula for the weight function  $((1 - t)^\alpha)((1 + t)^\beta)$  on
% the interval  $-1 < t < 1$ . The Gauss-Christoffel weights are  $m(0)$  times the
% squares of the first elements of the normalized eigenvectors of T, where
%  $m(0) = b(0)^2 = B(\alpha + 1, \beta + 1)2^{(\alpha + \beta - 1)}$  is the total mass.
% B is the beta function. The weight function is positive and integrable if
%  $\alpha + 1 > 0$  and  $\beta + 1 > 0$ .
%
% mxtj(n,alpha) takes  $\beta = \alpha$ . p is the nth ULTRASPHERICAL polynomial,
% with weight function  $(1 - t^2)^\alpha$  on the interval  $-1 < t < 1$ . Special
% cases are the CHEBYSHEV polynomial of the FIRST KIND, with  $\alpha = -1/2$ ,
% and of the SECOND KIND, with  $\alpha = 1/2$ .
%
% mxtj(n) takes  $\alpha = \beta = 0$ . p is the nth LEGENDRE polynomial, with
% weight function  $w(t) = 1$  on the interval  $-1 < t < 1$ . The quadrature
% formula here is originally due to Gauss. Christoffel generalized Gauss'
% formula to a wide class of weight functions. Because of this the Gauss-
% Christoffel weights are usually called Christoffel numbers.

```

% Copyright (c) 2 February 1991 by Bill Gragg. All rights reserved.

% mxtj calls mxt.

% begin mxtj

if nargin < 2 alpha = 0; end; if nargin < 3 beta = alpha; end

a = alpha; b = beta; c = a + b; d = b - a;

s(1) = d/(c + 2); t(1) = (a + 1)\*(b + 1)/(c + 2)^2/(c + 3);

if n > 2

    d = c\*d;

    n = (2:n)'; m = 2\*n; mm = m - 1; mp = m + 1;

    s(n) = d./(c + m)./(c + (m - 2));

    t(n) = n.\*(a + n).\*(b + n).\*(c + n)./(c + mm)./((c + m).^2)./(c + mp);

end

a = s(:); b = 2\*sqrt(t(:));

if nargout < 2 a = mxt(a,b); end

% end mxtj



## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center ..... 2  
8725 John J. Kingman Rd., Ste 0944  
Ft. Belvoir, VA 22060-6218
  
2. Dudley Knox Library ..... 2  
Naval Postgraduate School  
411 Dyer Rd.  
Monterey, CA 93943-5101
  
3. Department Chairman, Code AA ..... 1  
Department of Aeronautics and Astronautics  
Naval Postgraduate School  
Monterey, CA 93943-5000
  
4. Department of Aeronautics and Astronautics ..... 2  
ATTN: Professor I. Michael Ross, Code AA/Ro  
Naval Postgraduate School  
Monterey, CA 93943-5000
  
5. Space Systems Academic Group ..... 1  
Chairman, Code SP  
Naval Postgraduate School  
Monterey, Ca 93943-5000
  
6. Department of Mathematics ..... 1  
ATTN: Professor Fariba Fahroo, Code Ma/Ff  
Naval Postgraduate School  
Monterey, CA 93943-5000

- 7. Department of Mathematics ..... 1  
 ATTN: Professor D. A. Danielson, Code Ma/Dd  
 Naval Postgraduate School  
 Monterey, CA 93943-5000
  
- 8. Mr. Justin Comstock ..... 1  
 Welkin Associates, LTD.  
 10300 Eaton Place  
 Suite 410  
 Fairfax, VA 22030
  
- 9. Mr. Timothy Dawn ..... 1  
 Lyndon B. Johnson Space Center  
 NASA Code EA 44  
 2101 NASA Road 1  
 Houston, TX 77058
  
- 10. Space Systems/LORAL..... 1  
 ATTN: Lee A. Barker  
 3825 Fabian Way M/S G-76  
 Palo Alto, CA 94303-4604
  
- 11. Lieutenant Karl Jensen ..... 2  
 214 Newhall Place  
 Leesburg, VA 20175